

# Chapter 1. Basic Concepts

## Table of Contents

Objectives .....	2
1.1 Introduction .....	2
1.2 Hypertext .....	2
1.2.1 Anchors and Links .....	3
1.2.2 Jumps .....	3
1.2.3 Knowledge Additivity.....	4
1.2.4 Chain of Links .....	4
1.2.5 Loops and Mesh.....	4
1.2.6 Hypermedia.....	4
1.2.7 Authoring Hypertext .....	4
1.2.8 Getting lost in 'hyperspace' .....	5
1.3 The Ultimate Hypermedia System: The World Wide Web .....	5
1.3.1 Basic Ideas of the Web .....	5
1.3.2 Fields of Application .....	6
1.3.3 The Web as a Digital Library.....	6
1.4 Summary of Web Terminologies.....	7
1.4.1 Network Protocols .....	7
1.4.2 Web Application (Webapp).....	7
1.4.3 Uniform Resource Locator (URL).....	8
1.4.4 HyperText Markup Language (HTML).....	8
1.4.5 HyperText Transfer Protocol (HTTP).....	8
1.5 The Client-server Computing Model.....	9
1.5.1 A Definition and some History .....	9
1.5.2 Functionality .....	10
1.5.3 Information and Processing on the Web.....	11
1.5.4 MIME Types.....	11
1.5.5 Web Servers.....	12
1.5.6 Distributed Processing .....	13
1.6 Review Questions .....	14
1.7 Answers to Exercises .....	15
1.7.1 Exercise 2.....	15
1.7.2 Exercise 3.....	16
1.7.3 Exercise 4.....	16
1.7.4 Exercise 5.....	16
1.7.5 Exercise 6.....	17
1.7.6 Review Question 1.....	17
1.7.7 Review Question 2.....	17
1.7.8 Review Question 3.....	17
1.7.9 Review Question 4.....	17
1.7.10 Review Question 5.....	17
1.7.11 Review Question 6.....	18

# Objectives

At the end of this chapter you will be able to:

- Explain the basics of Hypertext;
- Explain web technology terminologies;
- Understand the client-server computing model.

## 1.1 Introduction

In 2009 the Internet celebrated its 40th anniversary, and the World Wide Web had been in existence for over 15 years. The concepts of computer networks and hypertext on which these technologies rely are only a little older. And yet the speed of development of these technologies, the speed of uptake by companies, and the speed of acceptance by consumers is unlike anything mankind has witnessed. Although both the Internet and the Web are firmly rooted in academic, altruistic endeavour, there is no doubt that the commercial interests are currently driving much of the technological development. This module aims to prepare you for contributing to this endeavour by helping you to understand the basic ideas and technologies behind the Internet, and giving you the opportunity to design and write Web pages using HTML5 and JavaScript.

The module starts here with, inevitably, the more theoretical aspects of the Internet and the Web. We begin by explaining hypertext before moving on to the most elaborate hypermedia system, the Web, and the ideas of client-server computing that allow us to use it.

## 1.2 Hypertext

Take a dictionary and observe how its content is linked together. How do you search for the meaning of a word? How can you find another word synonymous with that word? The dictionary is a paper example of a hypertext system. So are encyclopedias, product catalogues, user help books, technical documentation and many other kinds of books. Information is obtained by searching through some kind of index - the dictionary is arranged in alphabetical order, and each word is its own index. Readers are then pointed to the page of any other related information. They can read the information they are interested in without having to read the document sequentially from beginning to end.

Hypertext systems allow for non-sequential or non-linear reading. This is the underlying idea of a hypertext system. The result is a multidimensional document that can be read by following different paths through it. In this section we will look into the application of hypertext in computer systems, mainly the World Wide Web hypertext system.

The main use of hypertext is in information retrieval applications. The ease of linking different pieces (fragments) of information is the important aspect of hypertext information retrieval. The information can be of various media: it may be fragments of textual documents, structured data from databases, or list of terms and their definitions. Any of these, or a mixture thereof, can make up the contents of a hypertext document.

Therefore, in a hypertext system it is possible to:

- link with a term that represents aspects of the content of a document;
- connect two related documents;
- relate a term to a fragment containing its definition and use; and
- link two related terms.

Such a hypertext system can store a large collection of textual and multimedia documents. Such a hypertext system gives the end-user access to a large repository of knowledge for reading, browsing and retrieving. This is a "database" of sorts, and is the reason why such a hypertext system is called a digital library. The Web started as an extensively large digital library. As it has grown in popularity, it has offered the possibility of interactive applications and commerce on the Internet, making it much more than a digital library.

### To do

Read about networked hypertext and hypermedia in your textbooks.

We now explain some basic concepts on the use of hypertext.

### 1.2.1 Anchors and Links

A hypertext document contains links referring to other parts of the document, or even to whole other documents. A hypertext document does not have to be read serially; the fragments of information can be accessed directly via the links contained in the document.

The links embedded in a document are known as hyperlinks. When selected, these hyperlinks allow for the portion of the document linked to by the hyperlink to be displayed. This allows the reader to jump to another part of the same page, another page in the same document, or another document. By following a series of hyperlinks, the reader can follow their own path through the document.

A computerised hypertext system implements this idea by including anchors and links in documents, which are usually represented by files. An anchor is a fragment of information which links to another document or portion thereof. It is the visual representation of a link. A link is the actual reference (or "pointer") to the other document. For example, in Figure 1 the fragment of Document A containing 'You can find this in Section 5 of B' is an anchor from which there is a link to the relevant section in Document B.

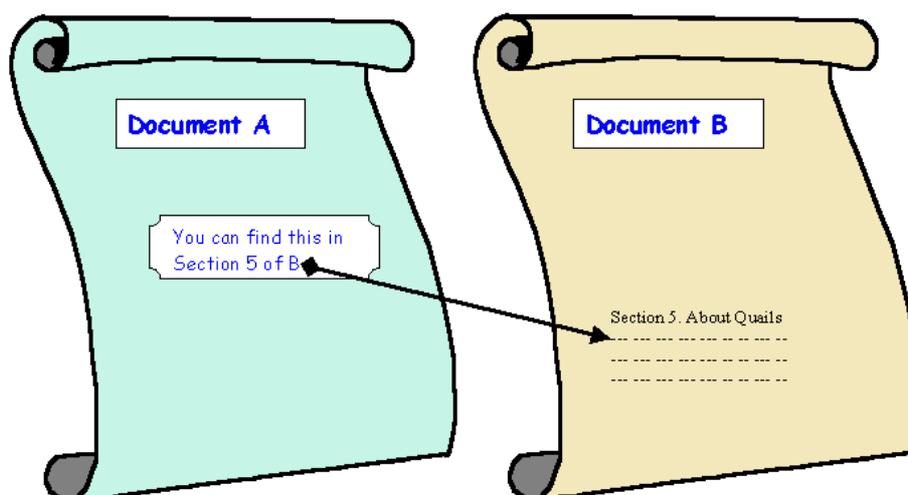


Figure 1: Illustration of Anchor and Link

Take care not to confuse anchor or link. A link is a pointer to another piece of information within the same document or in another document; often you cannot see how that link is implemented (it may be a hidden URL or some other programmed mechanism). An anchor is a fragment of information which the user interacts with in order to access the link. For instance, in a Web Browser the phrase "Click here to return to the previous page" is the anchor which the user interacts with — it contains the link to the previous page.

A hyperlink must have unambiguous reference to the document: this is usually information on the document's location (where in some file space or network it is) and the mechanism to access it (called the communication protocol). In Chapter 3 you will be introduced to HTML anchors and how the referenced documents are identified and located with URLs.

### 1.2.2 Jumps

A hypertext document allows links to portions of the document occurring before the link's anchor. This allows the reader to loop to portions of the document that they have already seen.

The table of contents at the beginning of this chapter is a collection of anchors with explicit links to the internal parts of the chapter. A book's bibliography is another collection of links but it refers to external information. To refer to the internal parts of this chapter is simple: the reader can merely turn to the appropriate page, identified by page number. However, referring to the external information given in a bibliography requires a more complicated effort of searching.

In computer-based hypertext documents, the mechanism to follow a link (the jump) is automatic. Jumping to an external link (another document) is as easy as jumping to an internal link within the same document. As long as the link is sufficiently specified with the name and the exact location of the linked document, the user can directly access the linked document with a simple click on the anchor.

## Basic Concepts

### 1.2.3 Knowledge Additivity

Links can be created to associate related subjects. Therefore the information given can be extensive and wide. The combination of two related subject areas is known as *knowledge additivity*.

Let's say you want to find out how to tailor a shirt using a sewing machine. You would probably look in a book on tailoring a shirt and another on using a sewing machine. The information read would then be linked together in your brain. However, with the hypertext concept, this knowledge additivity would be simpler with association links. You can just continue clicking to read on both subject areas within the perceived single document.

### 1.2.4 Chain of Links

A series of successive jumps constructs a chained path through a series of documents. There is no limit as to the number of jumps, therefore the size of the chain is not constrained.

There may be more than one link in a page and the reader is free to choose any of these links to follow. The path a reader takes will then be different from the path of another reader. Each sequence of jumps forms a different path to fragments of the overall information in the hypertext document. Generally, there is no rigid order to read the information in.

There are two different but complementary purposes of chaining documents via links:

- **Focusing:** each jump along the path, the user can narrow the scope of the search until the fragment containing the topic of their interest is reached.
- **Broadening:** Multiple outgoing links from a document allow the user to broaden their search. This is useful when the user does not have a precise idea of what is being searched for, or wishes to conduct a broad search in a certain domain.

Travelling through hypertext documents usually poses no technical difficulty. However, the reader might experience practical difficulties in retrieving a particular piece of information from a document with numerous alternative links.

### 1.2.5 Loops and Mesh

Just as the reader is free to choose which links and jumps a path through a hypertext document is to follow, it is possible for a user to return to a point previously visited. In other words, loops may exist. A path may even return to the original (home) document. Hence, the structure does not necessarily follow a linear pattern; instead, the documents are connected together in a graph / mesh defined by the links.

This critical property shifts the burden of devising suitable exploration paths from the designer of a hypertext document to the user. This changes the way information is stored and retrieved. Instead of searching directly for information, hypertext allows browsing for information. However, the mesh of information creates difficulty in navigating through the hypertext document.

### 1.2.6 Hypermedia

One of the original purposes for hypertext was the storage and management of textual documents. As computer and telecommunications technology has improved, the capabilities of hypertext systems have been extended to include any digitised media, such as sound and images.

This means that music and videos can be accessed via hyperlinks. This addition of multimedia to hypertext is known as Hypermedia. A combination of text, graphics, video or sound can now easily be interlinked in hypermedia document to offer a rich, often interactive, environment.

### 1.2.7 Authoring Hypertext

The process of preparing hypertext documents or, quite often, of converting a flat (linear) collection of documents into hypertext, is referred to as authoring.

Often an initial collection of documents has to be reorganised by splitting up the original documents into multiple sub-documents. Then links between these new documents must be constructed. Authors of hypertext documents are not only responsible for the content of these documents, but must link documents together, create paths through them, and build references that point to external documents associated to them.

Conceptually, related information is ultimately presented as a single, unique collection of hypertext documents. The remarkable aspect of hypertext or hypermedia documents that distinguishes them from other document types is that hypertext is 'shaped' by the user as he or she navigates the hypertext's network of link. Each sequence of links is a possible exploration path and each chosen sequence forms a single conceptual document for the user.

### 1.2.8 Getting lost in 'hyperspace'

The easy linking of different fragments of information crucial for browsing can produce hypertext documents that are very difficult to use. The user may become disoriented when they do not know where they are in the document and where he can go to. This problem of navigating a hypertext network is also known as being 'lost in hyperspace'. There are ways to minimise the risks of being lost in such a large information space.

#### **Return path**

The user simply backtracks through all the previous documents, link by link, until they reach the one they want to revisit. Alternatively, if the user remembers the reference of the required document, it may be selected from a list of the most recent documents explored.

#### **Home Page**

The starting fragment in a path is known as the home page. This home page is usually a well-defined document that contains the first links to a certain path. It helps to remind the user the path he has taken before and may even serve as a starting point to another path.

#### **Overview Diagrams**

This is the explicit display of the graph / mesh network of documents and links. Many websites have an overview site-map showing the paths the user may take to access certain information from the site.

#### **Guided Tours**

These are suggested paths arranged by the document's authors. Its purpose is to assist the user in the exploration of information in hypertext document. Tour documents form a logical path sequence by using simple 'next-document' or 'last-document' anchors.

#### **Direct jump**

This allows the user to move directly to a portion of a hypertext document. The user has to know the name and location of the portion to directly jump to it. In a Web browser, the *URL* address of the website is typed in and the requested page is retrieved and displayed to the user.

#### **Content-based retrieval**

Some documents may offer a search facility. Browsing for information through the search facility can help narrow the information space to the domain of interest. However, most current search facilities are restricted to textual information only.

## 1.3 The Ultimate Hypermedia System: The World Wide Web

### 1.3.1 Basic Ideas of the Web

The World Wide Web (Web) is a hypermedia system. It has largely achieved the goal of Tim Berners-Lee, its British inventor, of a universal information space. Tim Berners-Lee invented the World Wide Web in October 1994. Thanks to the global reach of the Internet, there is potentially universal access to an enormous volume of documents over the Internet. However, in many developing countries, access is poor, which raises issues of disenfranchisement and disempowerment. Many organisations make publicly available collections of hypermedia documents as part of either their marketing programme, customer service or global operations. Computer suppliers, for example, now publish very detailed specifications of their products via the Web.

Web servers and clients may be located at any part of the world and connected to each other by telecommunication links. If the Web is in some sense a digital library, it is one with no single geographical location. When it comes to commerce, distance begins to lose importance. As long as a supplier can provide goods or services where they are required, the location of the vendor and the consumer will not matter. This gives rise to issues about jurisdiction for taxes, consumer laws, legality of product, etc. This absence of distance is supported by the ease with which Web documents may be located world-wide; the mechanism is straightforward thanks to the way the location of such 'resources' are identified by a Uniform Resource Location (URL). The URL format unambiguously specifies locations of 'documents' on the Web. This location mechanism allows the actual implementation of geography-independent feature of the Web.

## Basic Concepts

Generally speaking, there is no central authority controlling the Web, although fully qualified domain names are subject to controlled allocation, and Internet Service Providers may be subject to the laws of the countries in which they operate. Furthermore, the World-Wide Web Consortium (W3C), headed by Tim Berners-Lee at the Massachusetts Institute of Technology, is influencing — and to a large degree controlling — how technologies are deployed on the Web. The W3C specifies HTML and XML, but other bodies, such as the European Computer Manufacturers Association (ECMA), have standardised other Web technologies, such as what we mostly call JavaScript. JavaScript is a programming language originally developed by Netscape.

Anyone with the appropriate knowledge, and with access to server space, can create a Web document. These Web documents can make reference to any other document. Moreover, a user does not require specific, proprietary software on their computer platform to access the Web, with many Web browsers being free software. While browsers can access and display the information on the Web, not all of them can supply the user with the interactive portions of the Web pages. For example, if Java applets are prohibited or a browser does not support JavaScript, interactivity with the Web document will be limited; some information may even be missing if that information required the presence of these interactive components.

The implications are easy to predict. With different browsers supporting different features, and with the navigation difficulties associated with hypertext's mesh / graph connections, chaos might ensue. However, even the most inexperienced users currently cope and the Web, and it is becoming both a universal world of information, and a universal place for doing business.

Dynamic pages can respond interactively to user input. It is possible to have portions of a hypertext document be produced by a programme as the document is requested. In this way, Web pages are increasingly being used as a front end to databases.

This allows the user to fill out a query and send it off for processing by the hypertext document. The server queries the database using the user's information and returns the output as HTML. To allow data to be sent in such a way to and from Web servers, a standard called the Common Gateway Interface (CGI) has been created. The difference between dynamic Web pages and non-dynamic (so called 'static') Web pages is transparent to the browser and user.

It is also possible to embed programmes inside HTML. When the browser loads such a page, the code is immediately executed. This mechanism supports remote transactions for the commercial aspect of the Web.

### 1.3.2 Fields of Application

The Web began as a tool to share knowledge and has successfully evolved into a general communications mechanism. With the support of transactions and synchronous communications, the Web has application in many different fields.

A primary use is the dissemination of knowledge, which takes many forms. For example, chat rooms and bulletin boards are integral to interactive discussion of all kinds of subjects. Frequently Asked Questions (FAQs) published on Web sites, offer answers to users' questions on how to do certain kinds of tasks. The variety of information that can be pulled out of the Web is wide-ranging.

Education includes a variation of the dissemination of knowledge. Open- or distance-learning programs spearhead this aspect of the Web. Basically, any kind of demonstration on how to carry out certain tasks can be considered education. For example, a user can learn how to create a Web page from the numerous websites publishing such instructions.

With the possibilities of online trading, business transactions are carried out on the Web. The user supplies their order and credit card details so as to buy products advertised on the Web. The Selling module would cover this subject area in depth.

### 1.3.3 The Web as a Digital Library

The Web as a vast digital library is becoming what is known as a 'Global Information Structure'. It will have a profound effect on how we live, work and play. We shall now look into a few of the social implications of the Web as a digital library and a marketplace.

#### i. Different Literacy

The hypermedia concept includes not only text and illustrations, but also music, animation, digital movies, video games and computer software. This diversity changes the form of literacy required when using the Web. The literacy needed when listening to music and watching a movie may be different from that used when reading a book. Less

## Basic Concepts

literacy may be required with innovative ways of using the digital library. For example, software that reads text aloud can assist people with visual handicaps.

### ii. Indeterminate Quality and Value

Editors and publishers employing traditional methods of publishing have little to gain from this type of publishing. As digital works can be copied at low costs, stored in almost no space and transported instantly anywhere in the world, writers can be their own publishers. Therefore, the works published are of indeterminate quality and value. Web publishing may provide no evaluation of work published.

### iii. Specialist Audiences

An article may perhaps interest a group of specialists in a particular field. With the Web, an average reader may browse through the article according to their degree of interest in the field. He or she may not want to be burdened with an additional flood of technicalities, or perhaps would navigate further to extract more in-depth information to supplement a deeper interest in the field

### iv. Copyright Issues and Ease of Purchasing

The ease of copying digital works causes difficulties in protecting copyrights. It may be tempting to make illegal copies rather than finding the rightful owners and paying them a fee. On the other hand, the non-issue of distance and the 24-hour, 365-day activity on the Web means that much can be easily bought through on-line shops. Consumers may come from distant areas or different time zones. With the Web, this market place is open at all times and can serve a very large global region. New technology even allows computational agents to staff the market place rather than people. Therefore, businesses are not constrained by distance or time.

### v. Sense of Place

Despite the irrelevance of distance, an electronic marketplace may be attractive as it *goes* to the consumers instead of them physically moving to the business environment. Its sense of place is created as an illusion for the benefit of the consumers.

### To Do:

Read more on the World-Wide Web in your textbooks.

## 1.4 Summary of Web Terminologies

Here we briefly summarise some of the terms you will need in the module; they will be studied to varying extents in later chapters.

### 1.4.1 Network Protocols

A network protocol is a standard way of regulating data transmission between computers. Just as diplomats adhere to protocols — rules of behavior — when in foreign lands, network communications do the same. They have to obey agreed rules if they are to communicate and 'get on with each other'. After many years of both public and private research and development, two network protocols are now dominant: TCP (Transaction Control Protocol) and IP (Internet Protocol), together known as TCP/IP. These were actually unlikely protocols to be so widely accepted, as faster, standardized protocols had been agreed upon, but none had the same robustness and extensibility as TCP/IP.)

Very often protocols were implemented without any formal acceptance and, because they worked most of the time, they became standards by default. Although TCP/IP is an accepted, de facto standard, work on Internet protocols continue in order to improve communication quality and support the continued growth of the Internet. There is no dictating authority for the Internet. Without a controlling authority, interim proposals about protocol changes are made by groups of interested individuals and then opened up for discussion. Documents containing the various proposed standards are published as Requests For Comment documents (RFCs). You may see references to a specific RFC as the best description of a protocol!

### 1.4.2 Web Application (Webapp)

A *web application* (or webapp), unlike standalone application, runs over the Internet. Examples of webapps are google, amazon, ebay, facebook and the UCT website. A webapp is typically a 3-tier (or multi-tier) client-server database application run over the Internet and it comprises five components:

## Basic Concepts

- HTTP Server: E.g., Apache HTTP Server, Apache Tomcat Server, Microsoft Internet Information Server (IIS), nginx, Google Web Server (GWS), and others. You will learn how to install Apache HTTP and Tomcat web servers in the next chapter.
- HTTP Client (or Web Browser): E.g., Internet Explorer (MSIE), FireFox, Chrome, Safari, and others.
- Database: E.g., Open-source MySQL, MariaDB, Apache Derby, mSQL, SQLite, PostgreSQL, OpenOffice's Base; Commercial Oracle, IBM DB2, SAP SyBase, MS SQL Server, MS Access; and others. You will learn how to install MySQL in the next chapter.
- Client-Side Programs: could be written in HTML Form, JavaScript, VBScript, Flash, and others. You will learn how to write client-side programs using HTML and JavaScript in this course.
- Server-Side Programs: could be written in Java Servlet/JSP, ASP, PHP, Perl, Python, CGI, and others.

### 1.4.3 Uniform Resource Locator (URL)

An URL is needed to locate any resources on the Web. It is an address format that specifies how and where to find a document. The general format is as follows, where the various items in italics must be substituted with part of a real URL, or omitted altogether.

```
http://machine_name:port/path/file_name.file_extension
```

**machine\_name** is either an IP address, for example 137.234.33.89, or a Fully Qualified Domain Name (also known as a DNS name, because Domain Name Servers map between Domain Names and IP addresses), for example, www.apple.com [http://www.apple.com]. In the machine name `http` is the protocol identifier, while `www.apple.com` is the resource name.

**port** is the TCP port to connect to; this is an entry point to software on the server; an optional part of a URL.

**path** is a relative file path from the server's document root; the server will start looking for a file in a specific directory and paths are relative to this

**file\_name** is the name of the file to be browsed, e.g. welcome

**file\_extension** is one of a number of suffixes which, by convention and operating system setup, indicate the type of data contained within the file, e.g. htm, html, txt. For example, in the URL below,

```
http://www.apple.com/retail/business/jointventure/terms.html
```

'terms.html' is a file with the html extension.

### 1.4.4 HyperText Markup Language (HTML)

This language provides the format for specifying simple logical structure and links in a hypertext document. As a markup language, special formatting commands are placed in the text describing how the final version should appear. These formatted documents are interpreted by a Web browser which uses the HTML code to format the page being displayed. Although most professionals use special authoring tools to write HTML documents and to manage sites, developers of e-commerce sites and applications need to know the nitty-gritty detail of HTML, and this is what you will study.

HTML has had several versions over the years. "HTML 2.0" was the first standard HTML specification which was published in 1995. HTML 4.01 was a major version of HTML and it was published in late 1999. Though HTML 4.01 version is widely used but currently we are having HTML 5 version which is an extension to HTML 4.01, and this version was published in 2012<sup>1</sup>. This course will take you through website creation using HTML5.

### 1.4.5 HyperText Transfer Protocol (HTTP)

HTTP is a network protocol used to retrieve documents from a variety of machines in a minimum amount of time. It was invented by Tim Berners-Lee to support a project in developing a distributed hypertext system. Distributed hypertext requires the retrieval of documents from many different machines. File Transfer Protocol (FTP), which predates the Web, would be too slow for this purpose as it is a connection-oriented protocol that requires a permanent connection to a server, thus requiring a connection-maintenance overhead when accessing different machines.

Therefore, to support browsing, HTTP has the following characteristics:

---

<sup>1</sup> [http://www.tutorialspoint.com/html/html\\_tutorial.pdf](http://www.tutorialspoint.com/html/html_tutorial.pdf)

## Basic Concepts

- **connection-less;** a connection is established only for the period of transfer, and the connection need not be maintained after thereafter;
- **stateless;** the server has no 'history' of client visits (although the implementation of cookies overcomes this);
- **comprehensive addressing;** diverse files on any HTTP server world-wide can be referenced via URLs;
- **diverse data;** using extensible MIME-types (see later), HTTP servers can supply information of every possible data types; and
- **rapid;** allows request-response cycles of less than 100 milliseconds

HTTP is not mandatory for distributed hypertext; there are other techniques and protocols that can be used to access or transfer information. However, like TCP/IP and HTML, HTTP is ubiquitous and so enables development in e-commerce.

### Exercise 1

Write down your ideas about the possible benefits of hypertext using the following headings. If you like, go on-line to discuss these with colleagues before writing them down.

- a) Ease of insertion of new information.
- b) Pointers to external materials.
- c) Browsing.

### Exercise 2

Write down your ideas about the possible drawbacks of hypertext using the following headings. If you like, go on-line to discuss these with colleagues before writing them down.

- a) Navigation Difficulties.
- b) No Main Catalogues.
- c) Network Overload.
- d) Link Fossilisation.

Discussions and answers can be found at the end of the chapter

### Activity 1: Analyzing Hyperlinks

Visit your favourite site and try to identify the following:

- a) A chain of links.
- b) A loop.
- c) A guided tour.

## 1.5 The Client-server Computing Model

When you are surfing the Web, you are using a Web browser. When you go to a website for documents, the site delivers them using software called the *Web server*. The browser is considered to be a client in the relationship with the server as it is requesting information services from the server. This is just one particular example of the client-server model of computing. You will learn how to setup your own web server using Windows and Ubuntu in the next chapter.

### 1.5.1 A Definition and some History

The client-server model has been defined as:

*A software partitioning paradigm in which a distributed system is split between one or more server tasks which accept requests, according to some protocol, from (distributed) client tasks, asking for information or action. There may be*

## Basic Concepts

*either one centralized server or several distributed ones. This model allows clients and servers to be placed independently on nodes in a network.*

Client-server computing is mainly about the client computer possessing its own computing power. In the days of mainframes, all the processing power took place on central computers. The client 'terminals' were little more than a television that could send and receive characters. When microprocessors became available, it was possible to make the terminals more powerful so that they could handle some of the processing. Over time this has meant that mainframes have been replaced by smaller server machines and terminals have been replaced by more powerful client workstations.

The client-server model provides a good division of processing power, since the server primarily provides information to the client, which is responsible for interpreting and displaying it. This means that servers do not have to be powerful machines, allowing more people to become service providers.

A more important characteristic is that because the client-server model provides for significant processing power at the (remote) client end, the operator of the client system has considerable autonomous power in contributing to the enterprise of which he or she is a part. This means that local decisions can be made possibly faster than if they were made remotely and action taken.

You may hear client-server computing being talked about as a modern computing 'paradigm'. Other than being part of a sales pitch, this is likely to mean that the model has made a significant impact on, and change to, the way we design and use computer systems. In particular, it is the current model for distributed business systems, and fits nicely into the emerging Web.

### 1.5.2 Functionality

In the context of the Web, users run client programs (i.e. Web browsers) which provide the following functionality:

- They allow the user to send a request for information to the server.
- They format the request so that the server can understand it.
- They format the response from the server in a way that the user can read it.

Server programs carry out the following:

- They receive a request from a client and process the request.
- They respond by sending the requested information back to the client.

In summary, the typical functionality of a client-server model is:

- A user, via a web browser (HTTP client), issues a URL request to an HTTP server to start a webapp.
- A client-side program (such as an HTML form) is loaded into client's browser.
- The user fills up the query criteria in the form.
- The client-side program sends the query parameters to a server-side program.
- The server-side program receives the query parameters, queries the database and returns the query result to the client.
- The client-side program displays the query result on the browser.
- The process repeats.

### Exercise 3

The client-server model applies to a lot of things outside of computers. Imagine going to a bank to withdraw some money? Who is the client and who is the server? Clearly, you are the client and the bank is the server. One of the advantages of the client-server model is that one server can handle many clients. The teller in the bank (server) handles many customers (clients). Also, you can use lots of different servers to get the service you need; that is there are a lot of tellers, and for that matter, bank branches and cash machines.

## Basic Concepts

For any website, say the University of Cape Town Computer Science website [<https://www.cs.uct.ac.za/>] or the University's Vula site [<https://vula.uct.ac.za/portal>], think about the following questions and write down your answers:

- a) Are there multiple clients?
- b) Who are these clients?
- c) Are there multiple servers?
- d) Why would there be multiple servers?

Discussions and answers can be found at the end of the chapter.

### 1.5.3 Information and Processing on the Web

Information is passed from the server to the browser. This information may be in the form of HTML documents, GIF files, Excel spreadsheets, movies — just about any digital content.

Information can also be passed from the browser to the server. When you click on a hyperlink you are sending information to the server, and when you fill in an online form, you are usually sending information to the server.

In addition to passing information backwards and forwards, some processing can also be done in the browser. For instance, you might have a simple Web page that calculates the overall cost of a loan once the initial value of the loan, the interest rate and the length of the loan have been entered.

But where does the processing take place? Does the server process the information and generate the result, or is it the client that processes the information? If the client does the processing, then this is a client-side application; if it is the server, it is a server-side application.

In the loan example above, the client has the information (the principle, rate and time). It could send this information to the server to process the information, generate the result and send it back to the client. Alternatively, the server could send a program to the client that will carry out the processing. In the latter case, since the client has all the information and the program is pretty small, it is probably better to run the application on the client side.

Of course, there is also a problem of who has the information. If the server has a database, and the client wants to query it, then there are two possibilities. The server could send the database and the querying program to the client to process it or the server could process it and simply send the result. In this case, it would probably be better to do the processing on the server side.

To summarize, where the processing is undertaken largely depends on where the information is, but it also depends on the processing loads of the machines as well as the size of the program being run.

#### Exercise 4

On the East Med. Trading Co. website, they would like to display to the user the number of pages that he or she has visited at that site. Think about the following questions and make a note of your answers.

- a) What data is needed?
- b) Where is the data stored?
- c) Should this be a client or a server side application?

Discussions and answers can be found at the end of the chapter.

### 1.5.4 MIME Types

A browser receives binary data from the server which it has to cope with. How does it know if the binary data is an HTML document, a GIF picture file or something entirely different? Even if it does know what kind of document it is, how does it process it? The answer to this is MIME types.

## Basic Concepts

MIME types — Multipurpose Internet Mail Extensions — were created to identify the differing types of possible email attachments. The MIME types have been extended to include new multimedia types as they have been introduced, and are now used with a variety of protocols including HTTP. When information is sent to a browser, a MIME header identifies the file type of the document. Attaching a MIME type to a file allows the browser to process the file's contents correctly without the browser having to guess at the data type from the file's extension. This is important, since while MS-DOS files require a three letter extension to identify a file type (and Windows XP uses a similar file extension system), not all operating systems do this.

This MIME header information has the following format:

```
Content-type: type/subtype
```

where

**type** is one of several general types such as: text, audio, image, video, application etc.

**subtype** is a more specific designation. This is a large and ever expanding category.

Some examples of MIME headers are

```
text/HTML video/MPEG  
image/GIF
```

### Processing MIME types

Mime types are processed as follows: Somehow the HTTP server must decide what type a file is. The server administrator can provide this information by instructing the server to map file extensions to certain file types. The server administrator must therefore supply a list of all the different file extensions for the files found on the server along with the equivalent MIME types for each of these file extensions.

The client browser must also be configured to know how to deal with these different types. Most browsers have been preconfigured, but they sometimes need to be updated to deal with new file types. On Google Chrome, for instance, the configurations of file type associations can be done under Settings. Files with different types can be viewed as follows:

View it in the browser. Files such a GIFs, JPGs etc. can all be handled by the browser

Use a plug in. Plug-ins are special pieces of code that software companies distribute to allow browsers to cope with new file formats. For example, plugins allow Google Chrome to use more features such as viewing animations using Flash.

Launch another application on your computer that can process the format.

If all else fails, the file can be saved to disk until a suitable program is found.

### Exercise 5

Suppose that there is a new document type that needs to be displayed on the client's computer and that you need to introduce a new MIME type for this document type. Let us use Microsoft Excel charts as an example (although this document type is obviously not new) and write down all the options to be considered on how this could be achieved.

Discussions and answers can be found at the end of the chapter

## 1.5.5 Web Servers

There are lots of different types of servers: FTP servers, gopher servers and, of course, Web servers. In this section we will concentrate on Web servers. What do Web servers need to do? Their functionality can be summarised as follows:

- They need to provide HTML pages (with an appropriate MIME type header).
- They need to provide other types of documents (also with an appropriate MIME type header).

## Basic Concepts

- They may need to process information from the user. For instance, if the user submits information to the site, the Web server must either process and store that information, or pass it on to another programme which can do so.
- They supply dynamic data (such as in response to user supplied information).

Processing user information and supplying dynamic data is complex. Many servers do not provide this facility. While complex to implement, it does make the server more dynamic and useful.

User information can be processed on the server using server-side applications called CGI (Common Gateway Interface) scripts. Many other languages and interfaces also exist, e.g. Java Servlets and PHP.

The server passes the user's data to the CGI program which then processes it. This program may dynamically create an HTML file to be sent back to the client just as standard HTML stored on the server would be. (Note, this will be discussed further in the units on JavaScript.)

In the next chapter, you will learn how to set up your own web server. Apart from learning how to set up a web server, you may wish to have a web server at home where you can store your files and then you can access and share these files with your colleagues at work.

### 1.5.6 Distributed Processing

Client-server computing is concerned with distributing the load of information and processing. Until about 20 years ago, most information was stored on one computer — the same computer on which all the processing was done. The only reason an extra copy of the data might have been kept on another machine was for security or backup purposes. If many people needed either the data or the processing, they would get another computer and copy the data.

With client-server computing, a given machine acts both as a client and as server; that is, it can run both a Web server and a browser client. It can also run processes (i.e. programs) on other machines. Network technology has enabled this distribution of processing and data.

The goal of distributing processing is to reduce the overall time that is needed to process some information. For example, consider this: one machine (named A) is connected to two other machines, B and C. If there are three processes to run, they can all run on A. If each machine requires 10 seconds of processor time in order to complete, then it will take a total of 30 seconds of user time to run the processes on one machine. But if B and C are each asked to run a process as well (so that now three machines are being used), then the total processing time has been distributed, and while it still takes 30 seconds of processor time to complete the work, it only takes 10 seconds of user time. It is therefore three times faster.

However, there is an additional cost that was overlooked in the previous paragraph. If A has to ask B to run a process, some communication time between the machines is required. For instance, just sending a message takes a certain amount of time, and this assumes that computer B already has the necessary data and programmes to run the process. If not, A may have to send the data and possibly the programme as well. Additionally, time is also required for B to send the results of the processing back to A. (The same is true for Machine C as well.)

For simplicity sake, let us say that sending the data and the results each takes one second. In the first second, A sends the data to both B and C, and A starts processing. In the following second, B and C begin processing. At the tenth second A finishes its processing. At second 11, both B and C finish processing their data and send their responses to A. In second 12, A receives the data and everything is completed. The total time to run the three processes is 12 seconds

Now try some process balancing in the following exercise.

### Exercise 6

The table below shows a list of processes (P1-P6) and computers (A-E) on which their data currently resides. Each process will output some result after a given amount of execution time has passed (as listed below). Processes can only execute on those computers which contain all of its data. The amount of data the processes require (in megabytes) is also given. Note that in some cases the data is already present on multiple computers. This data may be transferred to other machines at the rate of 1 MB per second. After the data has transferred, the process may then run on that machine. The computed results may also be transferred to another computer taking one second of time. All the machines are directly connected to each other and are otherwise identical. Each computer can run only one process at a time, but after a process completes may execute another.

## Basic Concepts

Of the five computers, computer A wants the results from four of the processes: P1, P2, P3 and P4. Computer B wants the results from P5 and P6, and computers C, D, and E are essentially idle, wanting no results from any of the processes.

Programme	Run Time	Location of Data	Size of Data
P1	5 seconds	A	8 MB
P2	6 seconds	D and C	4 MB
P3	7 seconds	A and C	5 MB
P4	8 seconds	C	12 MB
P5	9 seconds	A and E	2 MB
P6	10 seconds	B	2 MB

1. Come up with five different ways of distributing the processing, and the total user time for processing. For example:
  - a) machine A runs P1 (5 seconds).
  - b) machine B runs P6 (10 seconds).
  - c) machine C runs P3 (7 seconds plus one second to transfer the answer back to A for 8 seconds).
  - d) machine D runs P4 (12 seconds to send the data from machine C, 8 seconds for processing, and 1 second to send the answer for a total of 21 seconds).
  - e) machine E runs P5 (9 seconds to run and 1 second to transfer the answer to B for a total of 10 seconds).
  - f) After running P3, machine C also runs P2 (6 seconds plus one for transfer for a total of 7), bringing the amount of time that machine C is occupied for up to 15 seconds.
  - g) The longest time taken is for process P4, which takes 21 seconds to complete. This means that the total time for obtaining all the required results is only 21 seconds.
2. What is the least amount of time required to execute all six processes and send their results to the machines which want them? Is it possible to complete all of this work in less than 12 seconds?

Discussions and answers can be found at the end of the chapter

## 1.6 Review Questions

The answers to the review questions can be found at the end of the chapter.

- a) Is it relatively simple to insert new information in hypertext? \
- b) Is hypertext different from a hyper-document?
- c) Explain the reason why it is difficult to retrieve required information with unlimited chaining of information.
- d) Explain why Knowledge additivity enhances the learning process.
- e) Here is a summary of what hypertext and hypermedia are about. Fill in the blanks.

A hypertext document is a \_\_\_\_\_ document that implements anchors containing \_\_\_\_\_ to connect various fragments of information into one \_\_\_\_\_ network.

\_\_\_\_\_ is an extension to hypertext that includes digitised sounds and moving images.

The user is free to choose which links to follow in multi-linked hypertext documents. Each sequence of constructs a unique \_\_\_\_\_.

Authoring hypertext requires the decomposition of documents into fragments of information and then the construction of links between the\_\_\_\_\_.

The Web is an example of\_\_\_\_\_hypermedia. There is no central authority \_\_\_\_\_ its development and its information content is geographically-\_\_\_\_\_. The ease of linking information is one of the major benefits of hypertext.

Navigation difficulties through the various possibilities of different is the main drawback of hypertext. Extracting the required information can become tedious.

- f) What is so important about the client-server model of computing?

## 1.7 Answers to Exercises

### 1.7.1 Exercise 2

The following are some thoughts on the difficulties with working with hypertext

#### i. Navigation Difficulties

Navigation is the main drawback of hypertext. As the document is interlinked and may loop, readers can easily lose track of where they have been and where they are. The freedom to choose to follow any of the links may take the reader away from the item being sought. There are insufficient clues as to where an anchor links to.

#### ii. No Main Catalogues

A catalogue is readily available in a physical library and the user can easily find out whether the book he requires is available or not. It is difficult to index a hypertext document due to multiple links within a document, unless the reader is guided to a certain sequence of links.

Therefore, extraction of required information is tedious, especially in large hypertext documents. Search facilities attempt to make this easier. However, when searching a large content of information, such as the Web, the results of the search itself may be extensive. Moreover, these search facilities usually only recognise text descriptions and not the multimedia content of the document. This undermines the benefits of hypermedia.

#### iii. Link Fossilisation

Computer names may change and linked documents can be moved to other host computers. Hypertext requires explicit pointers to the names of the computers and the files on these computers, or the fragment of information that is linked to will no longer be accessible.

### iv. Network Overload

Hypertext content assumes universal coverage and infinite transfer capacity. The capacity of the telecommunications network may not be sufficient to cope with the usage without penalising or compromising other network activities. This happens when the reader is not warned of the document size, or is not conscious of the network implications. This results in a technical halt or slowdown in navigating through the hypertext.

#### 1.7.2 Exercise 3

- a) There are multiple clients for a website.
- b) The clients are all the people who visit it, or more precisely, they are the browsers used to view the site.
- c) There are no multiple servers for this site.
- d) For heavily used sites, the site is copied to another computer, called a mirror site. "Mirrors" are used to reduce traffic to the base site. Overall net traffic should also be reduced, as clients will go to a mirror that is closer than the base server. Mirrors add redundancy and make the site more likely to always be available. If one mirror goes down, other mirrors are likely to remain up.

#### 1.7.3 Exercise 4

The following might be needed to give a history of site usage.

- a) The data that is needed is a list of all the pages that a given browser has visited on the site.
- b) What the server can do is maintain a list of all the people who visit the site and which pages they have visited. This can be stored on the server.
- c) Alternatively, JavaScript code could be used to maintain the information on the browser using cookies. This is probably more difficult, and would only work for the current visit. (When the browser application was halted, all the information would be lost.) So, it would be better to store the information on the server side.
- d) If the data is stored on the server, then this should be a server-side application. The server can process the information and simply return a number representing the number of pages the user has visited. That number is returned to the browser. The alternative is to send all of the data (all the pages/user pairs) to the browser and then carry out the processing there.

#### 1.7.4 Exercise 5

The server will append the new type onto the information it sends. The server needs to know the type, but that should be fairly straightforward as the server is providing the data, so the administrator will have set that up. The client needs to know what the type is, and have a method of displaying the data. The browser itself could display the data, e.g. like a GIF file. The browser could use a plug-in. If it were a new file type, the group who developed the file type might provide a plug-in to read the type.

The other options are automatically invoked from current browsers. Since the browser does not know what to do with the file type, it puts up a dialogue box to ask the user. If the user knows which application to invoke, the user names that application that is then invoked and passed the file (the Excel chart file in this case). Instead of opening the file, you could execute Excel, open the file, save it, and deal with it later.

### 1.7.5 Exercise 6

Discuss the results of exercise 6 with your colleagues studying the module. In particular, you should be able to state the smallest amount of time which you were able to run all six processes in, and be able to explain your results.

### 1.7.6 Review Question 1

It is easy to create anchors which will link fragments of information together in a hypertext document. There is no real sequence to the information. New fragments can be inserted anywhere in a hypertext document as long as the anchors are properly implemented to link the new and existing fragments.

### 1.7.7 Review Question 2

Hyper-documents are hypertext documents. They are the same thing.

### 1.7.8 Review Question 3

If the hypertext document is small and does not contain many external links, information can be retrieved quickly with the browsing feature. The difficulty arises when the hyperspace is vast and there are many links in each page. Numerous links imply the possibilities of many different exploration paths. This makes navigation through the network of documents for the required information tedious.

### 1.7.9 Review Question 4

*Knowledge additivity* connects different aspects of information from different fields of study together, therefore the information is more useful. Let's say X=hunting skills and Y=using bows and arrow. Take a reader who wants to know how to hunt well with bows and arrows (Z). It is possible to achieve this with knowledge additivity in hypertext. ( $Z=X+Y$ ).

### 1.7.10 Review Question 5

A hypertext document is a non-linear document that implements anchors containing links to connect various fragments of information into one mesh network.

Hypermedia is an extension to hypertext that includes digitised sounds and moving images.

The user is free to choose which links to follow in multi-linked hypertext document. Each sequence of links constructs a unique navigation path.

Authoring hypertext requires the decomposition of documents into fragments of information and then the construction of links between the fragments.

The Web is an example of networked hypermedia. There is no central authority dictating its development and its information content is geographically-independent. The ease of linking information is one of the major benefits of hypertext.

Navigation difficulties through the various possibilities of different paths is the main drawback of hypertext. Extracting the information required can become tedious.

### **1.7.11 Review Question 6**

It is important because it allows for significant processing power at the remote client end so that the operator of the client system has considerable autonomous power in contributing to the enterprise of which he or she is a part.

# Chapter 2. Setting up a Web Server

## Table of Contents

2.1	Wampserver and Apache HTTP on Windows.....	1
2.1.1	Requirements.....	1
2.1.2	Installing Wampserver.....	2
2.1.3	Setting up Server Passwords .....	3
2.1.4	Testing Applications.....	5
2.2	Apache Tomcat on Windows .....	6
2.2.1	Requirements.....	6
2.2.2	Tomcat Setup.....	7
2.2.3	Testing Applications.....	10
2.3	Lampserver and Apache HTTP on Ubuntu 15.04 .....	10
2.3.1	Requirements.....	10
2.3.2	Installing Apache, PHP and MySQL.....	11
2.3.3	Testing Applications.....	13
2.4	Apache Tomcat on Ubuntu 15.04.....	13
2.4.1	Testing Applications.....	14

## Objectives

At the end of this unit you will be able to:

- install and setup Wamp server and Apache server on Windows;
- install and setup Tomcat Server on Windows;
- install and setup Lamp server and Apache server on Ubuntu 15.04; and
- install and setup Tomcat on Ubuntu 15.04.

## 2.1 Wampserver and Apache HTTP on Windows

In this section, you will learn how to set up a Web Server on a Windows PC. The steps in this section will illustrate how to use Apache HTTP. The next section will illustrate the setup for Apache Tomcat. Apache is a popular Web Server that allows users to easily set up their own Web Servers. It has the advantage of being open-source and hence is free to download. Apache is the basic software needed to support running of HTML and related content. Additional software, such as Tomcat, can be installed to complement the Web Server. Tomcat is a server that is meant to run applications written in Java and JSP (Java Server Pages).

Some popular options for deploying Apache, and optionally PHP and MySQL on Windows are Apache Lounge, XAMPP and Wampserver. Wampserver was used for this example. WAMP is an acronym that stands for “Windows, Apache, MySQL, and PHP”.

### 2.1.1 Requirements

To illustrate the steps below a Windows 7 64-bit computer was used. The Windows computer was connected to a local area network (LAN) that has Internet access. You also need to know the IP address of your

# Web Servers

computer. You can find your IP address by typing 'ipconfig' at a command prompt. Find the entry labeled 'Ethernet adapter Local Area Network' and take note of the IPv4 address.

It is recommended to disable the Windows Firewall before starting the Web Server setup. The steps below are for a fresh installation of Wampserver (assumes that Wampserver had not been installed before).

## 2.1.2 Installing Wampserver

Download WAMP from <http://www.wampserver.com/en/>. You will have the option of choosing 64-bit or 32-bit installation depending on your PC. This example uses the 64-bit installation. Locate the downloaded Wampserver file and click on it. This will open an installation wizard as shown in Figure 1. Follow the instruction wizard and leave the default settings as they are. After successful installation you will get the window shown in Figure 2. Leave the 'launch WampServer 2 now' box checked and click on 'Finish' button (in future you can start WampServer by clicking on your Start menu and clicking on its menu). On your toolbar, you should now see a 'W' shaped icon. On left-clicking this icon, you get the pop-up management console in Figure 3. Click on 'Start all services' and then check the 'W' icon on your toolbar. If the 'W' icon is green it means that all services are running. If it is red it means that no services are running. If it is orange it means that some services are running. If everything was installed correctly you should see a window such as the one in Figure 4.



Figure 1: Welcome window to WAMP setup

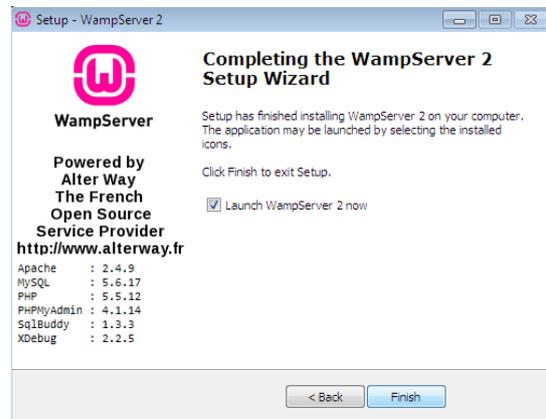


Figure 2: Finished installation

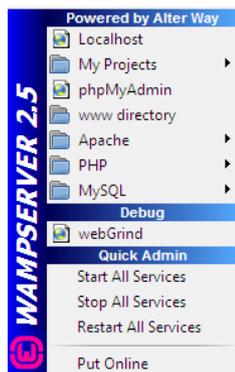


Figure 3: WampServer Management Console

# Web Servers

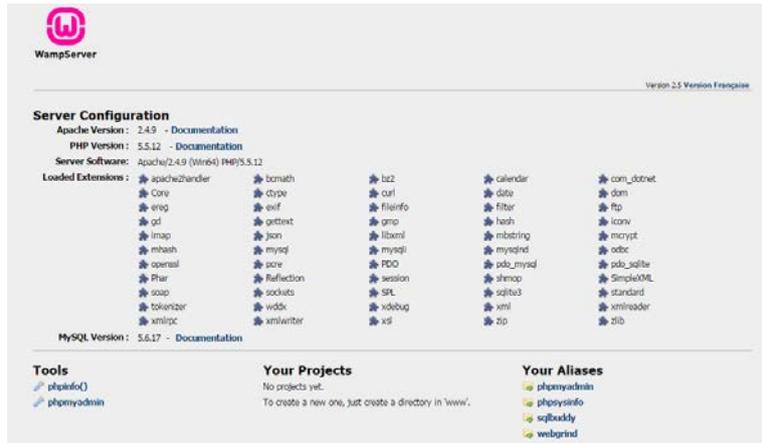


Figure 4: Localhost home screen

## 2.1.3 Setting up Server Passwords

In this step, you will learn how to create passwords for your server and also passwords to protect the files that you may want to share from your Web Server.

### i. Setting up MySQL and PHP Admin Password

On your local host home screen (Figure 4) click on 'phpmyadmin' under 'Tools'. The interface window opens showing the WAMP configuration page. At the bottom of this page, a message indicates that MySQL is running without a password (Figure 5).

Click on users and then check the box next to 'root localhost' and then click on 'Edit Privileges' (Figure 6). Scroll down to change the password and then click on 'Go' to save the changes. If you try to click on any other menu on the interface, for example on the SQL menu, you will get an error. Let us fix this by also changing the PHP admin password to match the MySQL password.

Open Windows Explorer. Navigate to the C:\wamp\apps\phpmyadminx.x.x\ folder (Figure 7). Inside that folder open **config.inc.php** – ideally using Notepad or any other html editor.

Search for the line `$cfg['blowfish_secret'] = ''`; – if you're using notepad it might be easier to just search for the word 'blowfish'. Change the line `$cfg['blowfish_secret'] = 'abcdef'`; to `$cfg['blowfish_secret'] = 'my passphrase'`; where **my passphrase** is your own password – **not the same one that you specified for root in MySQL**.

Now search for the phrase `['auth_type'] = 'config'`. Change 'config' to 'cookie'. Now search for `$cfg['Servers'][$i]['password'] = ''`; Replace the " with 'mysql-password'; where mysql-password is the MySQL password you created earlier.

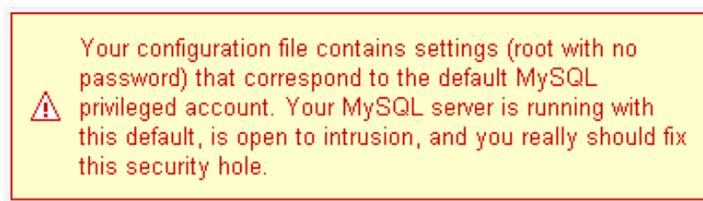
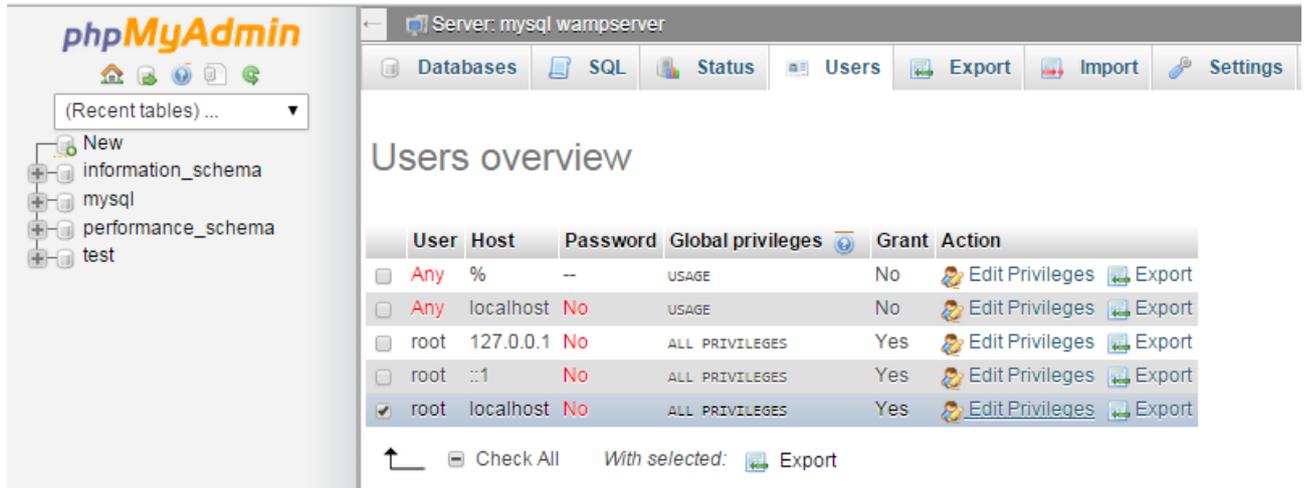
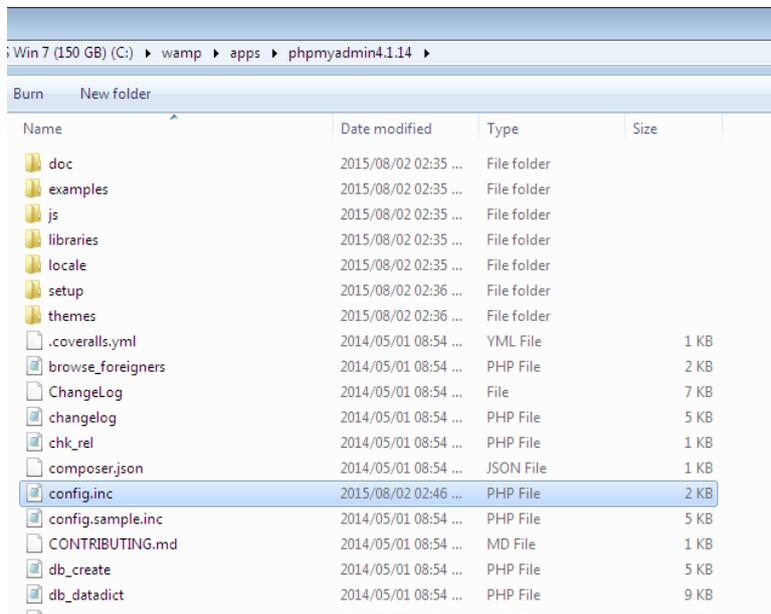


Figure 5: No Password set for WampServer

## Web Servers



**Figure 6: Changing root user privilege**



**Figure 7: Access config.inc folder**

Save the changes you have made and exit out of the editor. Go back to your toolbar where wampserver is located and click on 'restart all services'. Refresh localhost on your browser. Click on 'phpmyadmin'. You should now be prompted for a username and password. Your username is 'root' (since we did not change it from the default one) and your password is the MySQL password that you created.

### ii. Setting up a password to access files stored in the Web Server

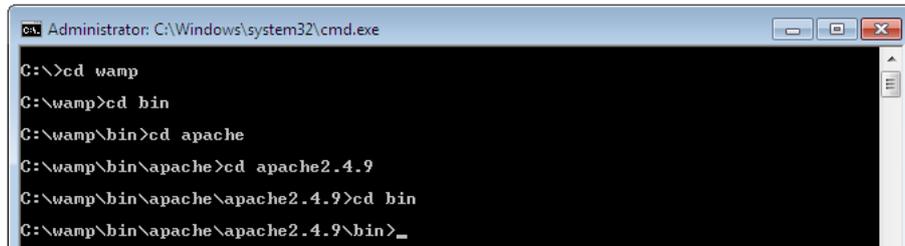
Now in case you want to share files from your server, you do not want just anyone to be able to access the files. So let us password-protect your files. For example, assume that you would like to store music files on your server and share them with others. First, decide where you would like to store your music files. In this illustration, the folder 'www' found in C:\wamp is used to store the music directory. We are assuming that you want your users to be able to access any folders that are stored inside the 'www' folder.

Next, using a command prompt, access the bin directory in the Apache folder (Figure 8). Then type:

## Web Servers

```
htpasswd -c "C:\wamp\my-password-file.txt" username
```

**my-password-file.txt** is the file where you create the password to access the 'www' folder. **username** is the username that you create to access the 'www' folder. Pick a username of your choice. **Note that the password file is not created inside the music folder.** After you press enter, you will be prompted to enter and re-enter a password. Create a password of your choice. This is the password that will be associated with the username you have just created, and the combination will be used to access the 'www' folder.



```
Administrator: C:\Windows\system32\cmd.exe
C:\>cd wamp
C:\wamp>cd bin
C:\wamp\bin>cd apache
C:\wamp\bin\apache>cd apache2.4.9
C:\wamp\bin\apache\apache2.4.9>cd bin
C:\wamp\bin\apache\apache2.4.9\bin>_
```

**Figure 8: Accessing Apache folder via command prompt**

Now we want to apply the login to your music folder. Open a new file in a plain text editor like Notepad. Copy and paste the following into it:

```
AuthType Basic
```

```
AuthName "This is a private area, please log in"
```

```
AuthUserFile "c:\wamp\my_password_file.txt"
```

```
AuthGroupFile /dev/null
```

```
require valid-user
```

You can replace the message **'this is a private area, please log in'** with your own security message that you would like users to see. Save this file in the 'www' folder which is our server root folder. There are two important things to note when saving this file:

1. Save this file as `.htaccess` (including the dot).
2. If using an html editor such as Notepad put quotation marks around the name, thus `".htaccess"`. This way, it will not be saved as a text file.

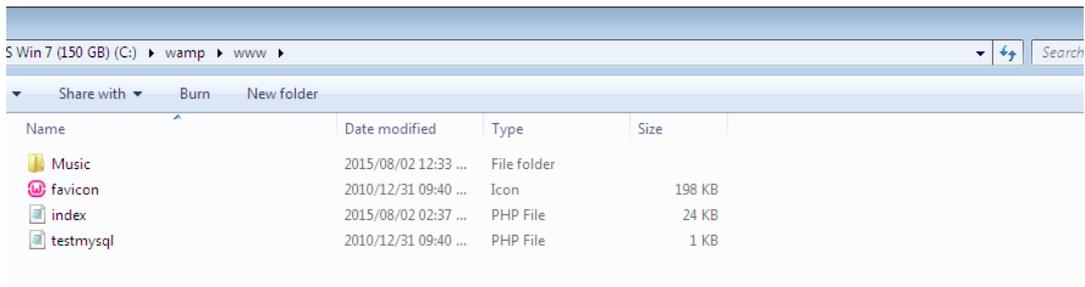
Now when you refresh your localhost on the browser, you should be prompted for login details.

### 2.1.4 Testing Applications

#### Sharing files stored on your WampServer

If you would like to share the music files that you have stored on your Web Server, simply create a directory called 'music' inside the 'www' folder (Figure 9). Put the music files that you would like to share in the 'music' folder. Type <http://localhost/music/> on your address bar and you should be able to see a listing of your music.

## Web Servers



**Figure 9: Music folder to be shared**

Take note of your computer's IP address. Go to your wampserver toolbar, left-click and click 'Put online'. Your server can now be accessed on the Web.

Share the address <http://xxx.xxx.xx.xxx/music/> with someone else to try on another computer. 'xxx' represents the numbers in your IP address. The user will be prompted for the login details that you setup in the last section. Note that this may not work if you are sharing the address with someone outside of your network who is behind a firewall.

We shall soon see how to write HTML files that can be accessed via the Apache HTML server.

The next section illustrates how to set up an Apache Tomcat server on Windows.

### Testing a 'Hello World' Application

Let us try a simple example to test that the websites we will create will work on this web server. Open Notepad and type the following code to print 'hello world' and save it as `hello.php` in the 'www' folder. Then access this file from your browser using your ip address or `localhost/hello.html`. You should get a 'hello world' output on your browser.

```
<html>
<head>
  <title>HTML Test</title>
</head>
<body>
  Hello world
</body>
</html>
```

## 2.2 Apache Tomcat on Windows

*Apache Tomcat* is a Java-capable HTTP server, which is able to execute special Java programs known as Java Servlet and Java Server Pages (JSP). It is also able to execute HTML files just like Apache HTTP.

### 2.2.1 Requirements

To illustrate the steps below a Windows 7 64-bit computer was used. The Windows computer had Internet access. Tomcat 8 was used for this installation. You need to have the latest Java JDK version installed. The recommended JDK version for Tomcat 8 is `jdk1.8`. Make sure that your computer is updated with this version. Go to [java.com](http://java.com) to download the latest version of Java.

The steps below are for a fresh installation of Apache Tomcat (assumes that Tomcat had not been installed before).

## 2.2.2 Tomcat Setup

### i. Download and Install Tomcat

Go to <http://tomcat.apache.org/> and download the latest version of Tomcat (Tomcat 9 at the time of writing this). Under Download on the left click on 'Tomcat 9' and under Binary Distribution click on 'Core' and you will see various packages. Click on the package applicable to you. For a Windows 64-bit computer click on '64-bit Windows zip'. Download this file.

Unzip the downloaded file to a directory of your choice. It is recommended not to unzip to the desktop as it is a difficult directory to locate from a command prompt. For this illustration a folder named 'tomcat' was created under drive D, hence D:/tomcat. The zipped file was extracted to this location. It is recommended to rename the unzipped file from the default name, for example, rename to 'apachetomcat' (Figure 10).

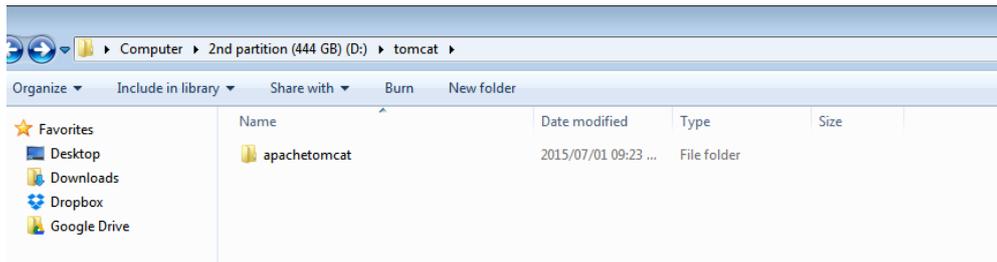


Figure 10: Creating folders to store tomcat files

### ii. Create an Environment Variable

First, take note of the directory into which JDK was installed. The default is "C:\Program Files\Java\jdk1.8.0\_{xx}", where {xx} is the latest upgrade number. It is important to verify your JDK installed directory before you proceed further. Start the command prompt and type 'set JAVA\_HOME' to check if the environmental variable had been set. If not, you will get the message 'Environment Variable JAVA\_HOME not set'. If JAVA\_HOME is set, check if it is set to your JDK installed directory correctly (For example in Figure 11). If not, proceed to set the environmental variable as below.

To set the environment variable JAVA\_HOME in Windows 7: Press "Start" button > Control Panel > System & Security > System > Advanced system settings > Switch to "Advanced" tab > Environment Variables > System Variables > "New" (or "Edit" for modification) > In "Variable Name", enter "JAVA\_HOME" > In "Variable Value", enter your JDK installed directory (e.g., "c:\Program Files\Java\jdk1.8.0\_51"). Click OK. To verify, **restart** the command prompt and type 'set JAVA\_HOME'. You should now see the output as in Figure 11.

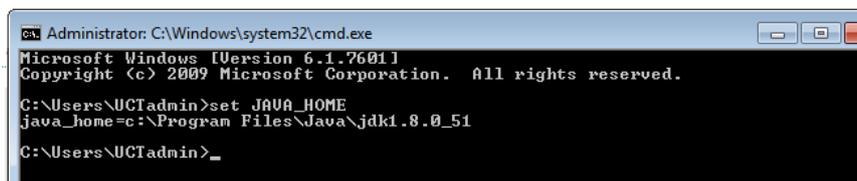


Figure 11: Checking environmental variables for JAVA\_HOME

### iii. Configure Tomcat Server

The Tomcat configuration files are located in the "conf" sub-directory of your Tomcat installed directory, e.g. "D:\tomcat\apachetomcat\conf". There are 4 configuration XML files: server.xml, web.xml, context.xml and tomcat-users.xml. Make a BACKUP of the configuration files before you proceed.

### Set the TCP Port Number

Use an HTML text editor (e.g., NotePad++) to open the configuration file "server.xml", under the "conf" sub-directory of Tomcat installed directory.

The default TCP port number configured in Tomcat is 8080, you may choose any number between 1024 and 65535, which is not used by an existing application. We shall choose 8888 in this article. Locate the following lines that define the HTTP connector, and change port="8080" to port="8888". Save the file and exit.

```
<Connector port="8888" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
```

### Enabling Directory Listing

Again, use an HTML text editor to open the configuration file "web.xml", under the "conf" sub-directory of Tomcat installed directory.

We shall enable directory listing by changing "listings" from "false" to "true" for the "default" servlet. Locate the following lines that define the "default" servlet; and change the "listings" from "false" to "true". Save and exit.

```
<servlet>
  <servlet-name>default</servlet-name>
  <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>0</param-value>
  </init-param>
  <init-param>
    <param-name>listings</param-name>
    <param-value>true</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

### Enabling Automatic Reload

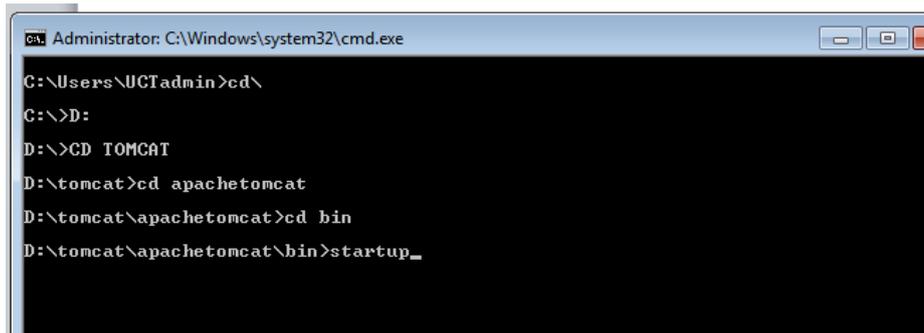
We shall add the attribute reloadable="true" to the <Context> element to enable automatic reload after code changes. Again, this is handy for test system but not for production, due to the overhead of detecting changes. Open context.xml and locate the <Context> start element, and change it to <Context reloadable="true">. Save and exit.

```
<Context reloadable = "true">
..... </Context>
```

#### iv. Start Tomcat Server

Launch a CMD shell. Set the current directory to the tomcat directory\bin", and run "startup.bat" as in Figure 12:

## Web Servers



```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\UCTadmin>cd\
C:\>D:
D:\>CD TOMCAT
D:\tomcat>cd apachetomcat
D:\tomcat\apachetomcat>cd bin
D:\tomcat\apachetomcat\bin>startup_
```

Figure 12: Starting tomcat from command prompt

A new Tomcat console window appears (look out for the Tomcat's port number (double check that Tomcat is running on port 8888). Future error messages will be sent to this console. Output messages from related Java programs are also sent to this console. If you want to shut down the server type 'shutdown' in place of 'startup'.

Start a browser. Issue URL "http://localhost:8888" to access the Tomcat server's welcome page. For users on the other machines over the net, they have to use the server's IP address or DNS domain name or hostname in the format of "http://serverHostnameOrIPAddress:8888". Note that this may not work if you are sharing the address with someone outside of your network who is behind a firewall. If everything is setup correctly, you should see the screen in Figure 13.



Figure 13: Tomcat home screen on local host

### Install Tomcat's Sample Web Application

Go to: <http://localhost:8888/docs/>

Click the link: "3. First web application"

Click "Example App" under contents on the left side of the screen.

Click on the link "here" to download their "Sample Application". (Download link:

<http://localhost:8888/docs/appdev/sample/sample.war> )

Save to this location: C:\Tomcat\apachetomcat\webapps

Give the Tomcat container a minute and it will automatically extract the WAR file and create a Web Application called "Sample".

Test your install: <http://localhost:8888/sample>

If you wish to create your own directory under webapps, choose a *name* for your webapp. Let's call it "myapps".

Go to Tomcat's "webapps" sub-directory. Create the following directory structure for you webapp "myapps":

1. Under Tomcat's "webapps", create your webapp *root* directory "myapps" (i.e., "tomcat\apachetomcat\webapps\myapps").
2. Under "myapps", create a sub-directory "WEB-INF" (case sensitive, a "dash" not an underscore) (i.e., "tomcat\apachetomcat\webapps\myapps\WEB-INF").

3. Under "WEB-INF", create a sub-sub-directory "classes" (case sensitive, plural) (i.e., "tomcat\apachetomcat\webapps\myapps\WEB-INF\classes").

Restart your tomcat server to pick up the changes. Then on your browser type <http://localhost:8888/myapps/>

You should see the directory listing of the directory "tomcat\apachetomcat\webapps\myapps", which shall be empty (provided you have enabled directory listing in web.xml earlier).

We shall soon see how to write HTML and PHP files that can be accessed via the Tomcat server.

### 2.2.3 Testing Applications

#### Sharing files stored on your Tomcat Server

If you would like to share the music files that you have stored on your Web Server, simply create a directory called 'music' insider the 'myapps' folder. Put the music files that you would like to share in the 'music' folder. Type <http://localhost:8888/myapps/music/> on your address bar and you should be able to see a listing of your music.

#### Testing a 'Hello World' Application on Tomcat Server

Let us try a simple example to test that the websites we will create will work on this web server. Open Notepad and type the following code to print 'hello world' and save is as hello.html in the 'myapps' folder. Then access this file from your browser using your ip address or <http://localhost:8888/myapps/hello.html>. You should get 'hello world' output on your browser.

```
<html>
<head>
<title>HTML Test</title>
</head>
<body>
Hello world
</body>
</html>
```

## 2.3 Lampserver and Apache HTTP on Ubuntu 15.04

In this section, you will learn how to set up a Web Server on Ubuntu 15.04. The steps in this section will illustrate how to use Apache HTTP. The next section will illustrate the setup for Apache Tomcat. LAMP stack is a group of open source software used to get web servers up and running. The acronym stands for Linux, Apache, MySQL or MariaDB, and PHP. Since I assume that your computer is already running Ubuntu, the Linux part is taken care of.

### 2.3.1 Requirements

To illustrate the steps below a 64-bit computer running Ubuntu 15.04 was used. The computer was connected to a local area network (LAN) with Internet access. You also need to know your server IP address. You can find out your IP address by right clicking the network icon in the notification area and clicking Connection Information, or by running *ifconfig -a* on the terminal. To log into your server, you will need to know the password for the "root" user's account. First, run the command below to update your server, before which you will be prompted to enter your root password.

```
sudo apt-get update
```

### 2.3.2 Installing Apache, PHP and MySQL

Run the commands below to install Apache2 Web Server and wait for completion of installation.

```
sudo apt-get install apache2
```

Run the commands below to start the Apache2 Web Server.

```
sudo service apache2 start
```

At this time, if you browse to the server using its IP address, you'll see Apache2 default page for Ubuntu as in Figure 14. This is how you also tell the server is up and functioning.

The next step is to install PHP as well as its module to enable PHP apps or web services to function. There are hundreds of PHP modules, but these few will get most web services started. To install PHP and other modules, run the commands below.

```
sudo apt-get install php5 php5-mysql php5-curl php5-gd php5-snmp php5-mcrypt  
php5-tidy php5-xmllrpc libapache2-mod-php5
```

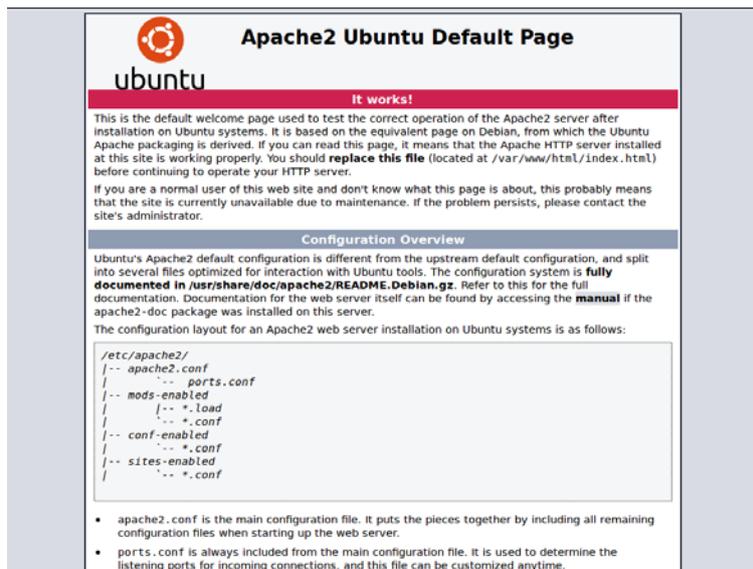


Figure 14: Successful Apache Installation

After installing the above modules, go to Apache root directory. It can be found at `/var/www/html` in Ubuntu. There create a file called `phpinfo.php`. Then in that file, add the lines below.

```
<?php  
phpinfo();  
>
```

Save the file, restart Apache and browser to the server IP address followed by `phpinfo.php`. (ex. <http://Your IP address/phpinfo.php>). There you'll find PHP information page such as Figure 15. This is how you also know PHP is functioning.

## Web Servers

Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-json.ini
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226.NTS
PHP Extension Build	API20131226.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	enabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:  
Zend Engine v2.6.0, Copyright (c) 1998-2014 Zend Technologies  
with Zend OPcache v7.0.4-dev, Copyright (c) 1999-2014, by Zend Technologies



**Figure 15: Successful PHP Installation**

The next step is to install MySQL. Run the commands below. Wait for completion.

```
sudo apt-get install mysql-server mysql-client
```

After installing the database server, you can start it using the commands below.

```
sudo systemctl start mysql
```

When it's started, run the commands below to configure the database server.

```
sudo mysql_secure_installation
```

When prompted, follow the options below and type your root password.

Next, choose Yes for the rest of the prompts until you're done.

- Enter current password for root (enter for none): Type root password
- Change the root password? **N**
- Remove anonymous users? **Y**
- Disallow root login remotely? **Y**
- Remove test database and access to it? **Y**
- Reload privilege tables now? **Y**

Restart Apache2 and you're done.

At this time, Apache2, PHP5 and other modules and MySQL database server should be installed and functioning. Your Ubuntu server is ready for any open source application that supports the LAMP stack.

You may wish to install PhpMyAdmin, which is a web interface through which you can easily manage/administer your MySQL/MariaDB databases. The installation can be completed with the following command:

```
sudo apt-get install phpmyadmin
```

Upon installation you will be asked to select the web server you are using. Select "Apache" and continue. Next you will be asked if you wish to configure phpmyadmin with dbconfig-common. Select "Yes". You need to perform one more step so that you can be able to access phpmyadmin from your Apache server. Run the following command.

```
sudo ln -s /usr/share/phpmyadmin /var/www/html
```

## Web Servers

If you now access phpmyadmin on your browser through <http://Your IP address/phpmyadmin> you should see the same window in Figure 16 and you can be able to log in using your MySQL username and password that you created.



Figure 16: Phpadmin home page

### 2.3.3 Testing Applications

#### Sharing files stored on your Tomcat Server

If you would like to share the music files that you have stored on your Web Server, create a directory called 'music' insider the 'var/www/html' folder.

Put the music files that you would like to share in the 'music' folder. Type <http://localhost/music/> on your address bar and you should be able to see a listing of your music.

#### Testing a 'Hello World' Application on Tomcat Server

Let us try a simple example to test that the websites we will create will work on this web server. Type the following command on the terminal to open the *gedit* editor.

```
sudo gedit
```

In the *gedit* editor, type the following code to print 'hello world' and save is at hello.html under var/www/html Then access this file from your browser using your ip address or localhost/hello.html. You should get 'hello world' as output in your browser.

```
<html>
<head>
<title>HTML Test</title>
</head>
<body>
Hello world
</body>
</html>
```

## 2.4 Apache Tomcat on Ubuntu 15.04

Make sure that Java JDK is installed on your machine. For this example, JDK-7 was installed. First, head-on-over to the Apache Tomcat 8 Download site. Then, under the heading 8.0.28 (the current version as of November 2015), or whichever is the newest version at the time you read this chapter, you'll see Binary Distributions. Under Binary Distributions you'll see Core and then under Core, you will see tar.gz. Right click on tar.gz and copy the URL link location.

## Web Servers

In the terminal use the URL you copied thus:

```
wget http://apache.is.co.za/tomcat/tomcat-8/v8.0.28/bin/apache-tomcat-8.0.28.tar.gz
```

After the download completes, decompress the file using the following command:

```
tar xvzf apache-tomcat-8.0.28.tar.gz
```

Now, move the file into a proper location using the following command. I am moving it to a folder called *opt*.

```
mv apache-tomcat-8.0.28 /opt
```

Now let's set the environment variables in *.bashrc* by typing the following command:

```
vim ~/.bashrc
```

Once in the vim editor, type 'i' in order to enter the insert mode. Add this information to the end of the file. After that press **esc** on the keyboard to go back to normal mode. Next press ':' (the colon) and this will drop the cursor to the bottom of the terminal. Here you save your changed by typing 'w' and press enter, and then you can type 'q' to quit vim.

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export CATALINA_HOME=/opt/apache-tomcat-8.0.28
```

Make the changes effective by running the following command:

```
./~/.bashrc
```

Tomcat should now be installed and configured for your server. To activate Tomcat, run the following script on the terminal. Note that the commands are case sensitive.

```
$CATALINA_HOME/bin/startup.sh
```

You can verify that Tomcat is installed correctly by typing <http://localhost:8080> in your browser. You should see the window indicating successful installation.

### 2.4.1 Testing Applications

#### Sharing files stored on your Tomcat Server

If you would like to share the music files that you have stored on your Web Server, simply create a directory called 'music' insider the 'webapps' folder. Put the music files that you would like to share in the 'music' folder. Type <http://localhost/music/> on your address bar and you should be able to see a listing of your music.

#### Testing a 'Hello World' Application on Tomcat Server

Let us try a simple example to test that the websites we will create will work on this web server. Type the following command on the terminal to open the *gedit* editor.

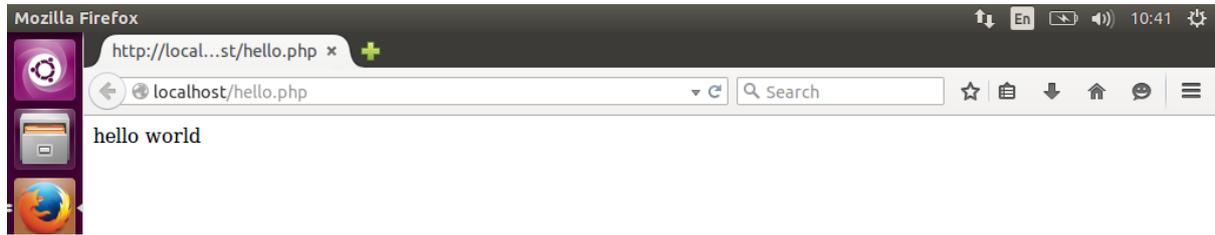
```
sudo gedit
```

In the *gedit* editor, type the following code to print 'hello world' and save is at *hello.html* under the *webapps* folder. Then access this file from your browser using your ip address or *localhost/hello.html*. You should get the output in Figure 17.

```
<html>
<head>
<title>HTML Test</title>
</head>
```

## Web Servers

```
<body>  
Hello world  
</body>  
</html>
```



**Figure 17: Hello World Output**

# Chapter 3. HTML: BASICS

## Table of Contents

Objectives .....	1
3.1 Basics .....	2
3.1.1 HTML Markup .....	2
3.1.2 Nesting HTML Tags .....	2
3.1.3 Creating HTML Text using Notepad++ .....	2
3.1.4 Standard HTML Document Structure Format .....	3
3.2 HTML Formatting .....	4
3.2.1 The Browser As Formatter .....	4
3.2.2 Paragraphs, Line Breaks and Preformatting .....	5
3.2.3 Headings, Horizontal Rules and Meta Tags .....	5
3.3 Lists .....	7
3.4 HTML Comments .....	9
3.5 Anchors .....	9
3.5.1 Linking to Email Addresses & other Non-Web Links .....	10
3.5.2 Linking to Sections within Documents .....	10
3.5.3 Targeting Windows .....	11
3.5.4 Link Appearance .....	12
3.6 Multimedia .....	12
3.6.1 Graphics .....	12
3.6.2 Objects .....	13
3.6.3 ImageMap .....	14
3.7 Writing Good HTML .....	14
3.8 Discussion and Answers .....	15
3.8.1 Discussion Topics .....	15
3.8.2 Discussion of Activity 2 .....	15
3.8.3 Discussion of Activity 4 and 5 .....	15
3.8.4 Discussion of Activity 6 - Lists .....	15
3.8.5 Discussion of Activity 7 – Comments .....	17
3.8.6 Discussion of Activity 9 – Linking to sections within a document .....	17
3.8.7 Discussion of Activity 11 – Changing appearance of links .....	17

## Objectives

At the end of this chapter you will be able to:

- Create HTML files using Notepad and run them on a Tomcat server;
- Use HTML tags to write HTML files;
- Format HTML files;

- Create lists in HTML files;
- Use anchors in HTML files;
- Use multimedia in HTML files.

# 3.1 Basics

## 3.1.1 HTML Markup

HTML pages are created by tagging textual information with HTML markup. HTML markup consists of tags, which appear inside angled brackets `<` and `>`

An example of an HTML tag is `<B>`, which causes text to appear in bold. `<B>` only notes where text should begin to appear in bold, while the tag `</B>` marks the end of the emboldening. Most HTML tags have a corresponding end tag, which is specified by the name of the tag preceded by the `/` character.

So, to create the text:

**Internet Commerce is great!**

The text is marked up as:

```
<B>Internet Commerce is great!</B>
```

Another example of an HTML tag is `<I>`, which causes text to appear in italic. In HTML 4.01, the `<I>` tag was used to render text in italics. However, this is not necessarily the case with HTML5. Style sheets can be used to format the text inside the `<I>` element. This will be demonstrated later.

Note that tags are not case-sensitive. In other words, `<B>` or `<b>` are the same tag, both specifying bold text.

## 3.1.2 Nesting HTML Tags

Text may be both bold and italicised. This is done by using both the `<B>` and `<I>` tags. When doing so, it is important to remember not to overlap HTML tags. In other words:

```
<B><I>Internet Commerce  
is great!</I></B>
```

is **correct**, but

```
<B><I>Internet Commerce is great!</B></I>
```

is **wrong**.

Overlapping tags is a common mistake. Although Web browsers are usually smart enough to work out what is meant, it can lead to problems. Furthermore, for an HTML page to be considered valid HTML, it must contain no overlapping tags.

### To Do:

Read the section on "HTML Tags" in your textbooks.

## 3.1.3 Creating HTML Text using Notepad++

This section covers the creation of an HTML page. You will need a Web browser and a text editor. Use

## HTML: Basics

any text editor you wish to, but the following Activity descriptions will use Notepad++. Notepad++ is a free Windows editor that also supports several programming languages. For example, you will notice that HTML keywords are highlighted in different colors.

1. Open your Web browser. This sections' goal is to create a Web document that can be opened with your browser.
2. Open Notepad++. It can be found by selecting Start, then All Programs, then Notepad++.
3. Type the following text into Notepad++: your name and the module number (CSC5003). Save this file as start.txt.
4. Now load start.txt into the browser by dragging start.txt onto your browser.
5. The browser should now display the text contained in **start.txt**. (If it does not, make sure that you have saved **start.txt** and that this is the file you are opening).
6. Once you have displayed start.txt, return to Notepad. Add the text "Internet Commerce", and save the file again.
7. Return to the Web browser and reload the document (by using either by using the Refresh or Reload toolbar buttons, or by selecting File/Open once again).
8. If you are able to see the new piece of text, you have successfully used Notepad to create your first Web page.

### Activity 1: Getting started with HTML

This Activity adds HTML tags to start.txt.

1. Open your file **start.txt** in Notepad.
2. Mark up the text "Internet Commerce" so that appears in bold. Do this by placing the `<B>` tag in front of the text, and `</B>` at the end of the text, as shown below:  
`<B>Internet Commerce</B>`
3. Save the file as **start.html**, since it contains some HTML formatting. Save the file with this new name (using Save As). Note that saving it as **start.htm** is also accepted. Other than the obvious, the letter "L," there's not much of a difference between the two extensions. Most, if not all, web browsers and servers will treat a file with an HTML extension exactly as it would a file with an HTML extension, and vice versa<sup>1</sup>.
4. Load **start.html** in the Web browser. **Internet Commerce** should now appear in bold.
5. Return to Notepad and add more text, some of it in bold and others in italics. (Remember `<I>` is the tag for italics) Save the document and reload it.

### 3.1.4 Standard HTML Document Structure Format

Although a number of HTML tags have been introduced that markup how text should be displayed in a browser, a correct HTML document must always include certain structural tags. These tags are `<HTML>`, `<HEAD>`, `<BODY>` and `<TITLE>`.

The `<HTML>` tag should be placed around the document's contents; this tells the browser that the whole document is written in HTML. Like a person, all HTML documents have only one head and one body. All the text of the HTML document should be inside either the head or the body. Roughly, the `<HEAD>` holds information about the document itself, and the `<BODY>` holds the information that should be displayed. The document's `<TITLE>` is given in the `<HEAD>`. The title is shown at the very top of the browser (i.e. in the title bar) — not in the browser window itself.

The standard structure of an HTML document is:

```
<HTML>
<HEAD>
<TITLE>Text to appear in the title bar of the browser</TITLE>
</HEAD>
<BODY>
```

---

<sup>1</sup> [http://www.sightspecific.com/~mosh/www\\_faq/ext.html](http://www.sightspecific.com/~mosh/www_faq/ext.html)

## HTML: Basics

```
    The text to appear in the main browser window.  
</BODY>  
</HTML>
```

This format should always be used when writing HTML documents.

**Note:** students are often confused about the use of the <BODY> tag, and they often include multiple body tags. This can lead to problems later on, so make sure to use only one <BODY> tag.

**To Do:** Read the section on HTML document structure in your textbooks.

### Activity 2: Structuring your HTML document

In this Activity you will convert your file that contains a few HTML tags into a correctly structured HTML document. Open start.htm in Notepad.

1. Add the <HTML> tag on the first line of the file (before anything else).
2. Add the </HTML> end tag on the last line of the file (after everything else).
3. Add the document header by adding a <HEAD> tag on the line underneath the <HTML> tag and the </HEAD> tag on the line beneath that.
4. Between the opening and closing <HEAD> tags, add the <TITLE> and </TITLE> tags.
5. Enter the text "My first Web page" between the <TITLE> tags.
6. Underneath the </HEAD> tag, create the body of the document by entering the <BODY> tag.
7. At the bottom of the document, add the </BODY> tag just before the </HTML> tag.
8. Save the file.

If you have problems correctly formatting the file, look at the code in 3.1.4.

You are probably thinking that it looks the same as the previous document. However, if you look closely at the title bar you should see that it now displays the words "My first Web page". The main difference, however, is that the browser now has to do a lot less work to do, since the document informs it of the HTML's structure.

### Activity 3: Loading your HTML file on Tomcat

The previous chapter guided you through tomcat installation. Let us launch the start.html file using the tomcat webserver. Make sure that your tomcat server has been started. Save start.html in the folder myapps that you created within the webapps folder. Load **start.html** in your browser by typing `http://localhost:8888/myapps/start.html`

## 3.2 HTML Formatting

### 3.2.1 The Browser As Formatter

As you will recall, it is the browser that actually formats the HTML document. But when it displays text, where does it put the line breaks?

The browser automatically determines the position of the line breaks. It tries to fit all of the text into the current window so that the user does not need to do any horizontal scrolling. If the browser window changes size, the browser reformats the document.

It also ignores extra spaces. If there are two spaces between words in the HTML file, the browser will display the text in exactly the same way as if there was only one. Blank lines are ignored in a similar way. The browser also tries to correct errors in incorrect HTML (such as HTML containing overlapping tags). When doing so, the browser may incorrectly interpret the HTML document, making it a wiser choice to write correct HTML. Sometimes it can be difficult to have the browser format things as you want. You will learn more tricks on how to do this as you work through the HTML units.

### 3.2.2 Paragraphs, Line Breaks and Preformatting

The tag `<BR>` is used to start a new line. `<BR>` is a standalone tag, that means there is no closing `</BR>` tag. Note that `<BR>` does not place a line space between the two lines. To do that you need to use the `<P>` paragraph tag. Do not forget to add the end tag `</p>` although most browsers will display HTML correctly even if you forget the end tag. The tag `<pre>` defines preformatted text. The text inside a `<pre>` element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks.

#### Activity 4: Paragraphs, Line Breaks and Preformatting

In this Activity you will use the `<P>` and `<BR>` tags to create line breaks in text. We will also demonstrate the use of `<pre>`.

1. Load Notepad and begin a new HTML document.
2. Enter the usual structural HTML tags. Set the title to "Formatting text".
3. Within the body type in the following text exactly as it appears below. Note how 'This is cool' has been typed. Do not use any HTML tags to format it at this stage.

Users of HTML are sometimes surprised to find that HTML gives them little control over the way that a page is displayed. It should be remembered that HTML was developed as a means of marking up the structure of a document not as a way of determining its presentation. Formatting text to appear on a Web page is therefore different from formatting text to appear in a printed document.

This

is

Cool.

4. Save the document as **format.html** in your myapps folder and load it in your browser to view it. Note that 'This is cool' is displayed without the line breaks.
5. Resize your browser and watch how the text is reformatted to fit in the resized browser window.
6. Return to Notepad and make the changes as shown in Figure 3.1.
7. Save the file again and load it in your browser to check your HTML. Resize the browser and watch how the document is reformatted for the resized window.

### 3.2.3 Headings, Horizontal Rules and Meta Tags

The DOCTYPE declaration defines the document type to be HTML. In HTML5 this is written as `<!DOCTYPE html>`. The `<!DOCTYPE>` declaration helps the browser to display a web page correctly. There are different document types on the web. To display a document correctly, the browser must know both type and version. The doctype declaration is not case sensitive. All cases are acceptable:

Another set of HTML tags are the headings tags. These are `<H1>`, `<H2>`, `<H3>`, `<H4>`, `<H5>` and `<H6>`. The text surrounded by the `<H1>` tag is displayed in a very large font size. Text surrounded by the `<H2>` tag is

```

1 <html>
2 <head>
3 <title> Formatting text </title>
4 </head>
5
6 <body>
7
8 Users of HTML are sometimes surprised to find that HTML gives them little control over the way
9 that a page is displayed. <br> It should be remembered that HTML was developed as a means of marking
10 up the structure of a document not as a way of determining its presentation. <p> Formatting text to
11 appear on a Web page is therefore different from formatting text to appear </p> in a printed document.
12 <pre>
13 This
14 is
15
16 cool.
17 </pre>
18 </body>
19 </html>

```

Figure 3.1: Tags for paragraph, line breaks and preformatting

## HTML: Basics

displayed in a slightly smaller font size, and so on down to the <H6> heading tag. You can use these tags to provide your page with a standard outline format. For example, the page heading might be displayed using the <H1> tag, a section heading using <H2> and a sub-section heading using <H3> and so on. Use HTML headings for headings only. Don't use headings to make text BIG or **bold**. Search engines use your headings to index the structure and content of your web pages. It is important to use headings to show the document structure. Browsers automatically add some empty space (a margin) before and after each heading.

Earlier we noted that Web browsers are HTML readers. Each browser is free to interpret HTML any way it likes. Consequently, a document read in one browser might look a little different to one read in another browser. Although the HTML standard states that <H1> tags should be as big as or bigger than <H2> tags, and <H2> tags should be as big as or bigger than <H3> tags and so on, one browser might display the <H3> tag with the same font size as the <H2> tag, while another browser might display it in a smaller font size. Hence the difference in displaying the same text. In practice, these implementation questions will become an issue when you are using more complex tags. For now you can ignore this problem while writing HTML.

The <hr> tag creates a horizontal line in an HTML page. The <hr> element can be used to separate content.

The HTML <meta> element is also meta data. It can be used to define the character set, and other information about the HTML document. Other meta elements that can be used are <style> and <link>.

### Activity 5: Headings

In this Activity you will set up a page heading and sub-heading for the Web page begun in Activity 5 and use the HTML headings tags to implement it.

1. Load format.htm in MS-Notepad.
2. Within the <head> tags, add <meta charset="UTF-8">. It does not matter whether it is below or after the <title> tag.
3. Set up the page heading "Formatting text" and place the <H1> heading tags around it, in other words, <H1>Formatting text</H1>.
4. Reload format.html in your browser. You will notice that the effect of the <H1> tag is to display the text not only in an enlarged font size but also to include extra space above and below it. So you do not need a <BR> or <P> tag as well.
5. Return to Notepad and use the <H2> tag to create a sub-heading for the page, "Paragraphs and line breaks".
6. Add <hr> between 'This' and 'is'.
7. Reload the document in your browser to check the HTML and you should have an output like in Figure 3.2.

#### HTML Tip - How to View HTML Source

Have you ever seen a Web page and wondered "Hey! How did they do that?" To find out, right-click in the page and select "View Page Source" (in Chrome) or "View Source" (in IE), or similar in another browser. This will open a window containing the HTML code of the page.

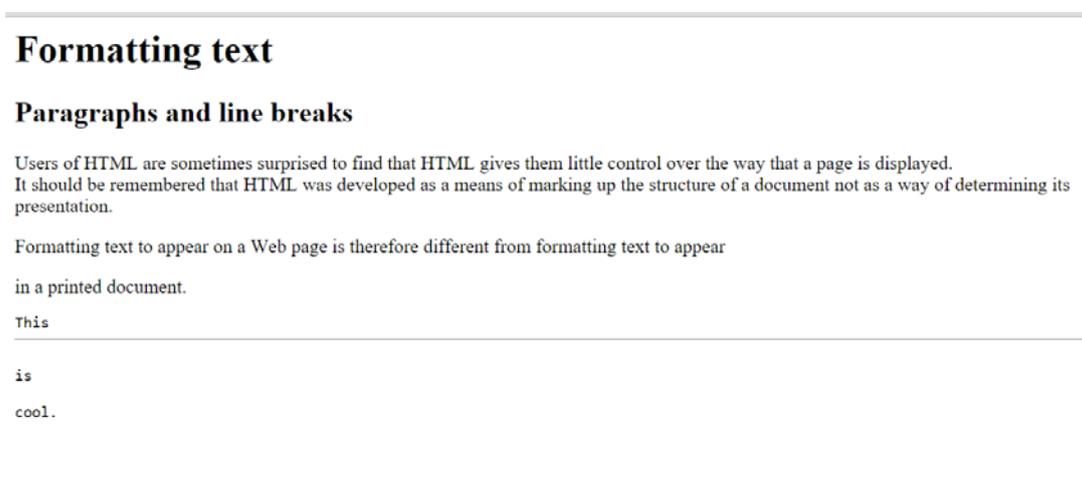


Figure 3.2: Using headings, horizontal rules and meta tags

## 3.3 Lists

- |           |            |
|-----------|------------|
| • Apple   | 1. Apples  |
| • Oranges | 2. Oranges |
| • Bananas | 3. Bananas |

The two examples above are lists. The list on the left uses bullets to mark the list elements, and is known as an **unordered list**. The list on the right uses numbers to mark the list elements and is known as an **ordered list**.

HTML lists consist of a list tag and list element tags.

In an **unordered list**, the list tag is `<UL>` and the list element tag is `<LI>`. Note that although the list element end tag `</LI>` was optional in previous versions of HTML, it no longer is. The list end tag `</UL>` is also not optional.

To create an **unordered list** as in the above example, use the following HTML.

```
<UL>
<LI>Apples</LI>
<LI>Oranges</LI>
<LI>Bananas</LI>
</UL>
```

Note that it is useful to indent the `<LI>` tags on the page to keep track of the level of indentation. To add more list elements, add extra list element tags `<LI>` `</LI>` containing the elements within the `<UL>` tags.

A **style** attribute can be added to an unordered list, to define the style of the marker:

Style	Description
<code>list-style-type:disc</code>	The list items will be marked with bullets (default)
<code>list-style-type:circle</code>	The list items will be marked with circles
<code>list-style-type:square</code>	The list items will be marked with squares
<code>list-style-type:none</code>	The list items will not be marked

For example the code below would append square bullets to the list.

```
<UL style = "list-style-type:square">
<LI>Apples</LI>
<LI>Oranges</LI>
<LI>Bananas</LI>
</UL>
```

**Ordered lists** are specified almost exactly the same as unordered lists, only the `<OL>` tag is used instead of the `<UL>` tag. A **type** attribute can be added to an ordered list, to define the type of the marker:

## HTML: Basics

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

For example,

```
<OL type = "i">
<LI>Apples</LI>
<LI>Oranges</LI>
<LI>Bananas</LI>
</OL>
```

The **description** list, is different: it has neither bullets nor numbers. The description list tag is `<DL>` and the list elements consist of a term and its definition. The term is marked by `<DT>` tags and the definition by `<DD>` tags. An example use definition lists is the glossary definition that appears below.

```
<DL>
<DT>HTML </DT>
<DD> Hypertext Markup Language; the format of Web documents </DD>
</DL>
```

Lists can be nested (lists inside lists). For example,

```
<OL type = "i">
<LI>Apples</LI>
<LI>Oranges</LI>
<LI>Bananas</LI>
  <ul>
    <li> small bananas </li>
    <li> big bananas </li>
  </ul>
</OL>
```

### Activity 6: Lists

In this Activity you will create a series of lists to practise your HTML list-building skills.

1. Load format.html in Notepad.
2. Underneath the text, create three lists as follows:
  - a. List one should be a circled bulleted (i.e. unordered) list, using square bullets, giving the days of the week.
  - b. List two should be a numbered list of the months of the year. Make the numbers lowercase roman numerals.

## HTML: Basics

- c. List three should be a definition list of the four seasons.
3. Save the file and view it in your Web browser to ensure that it displays as desired.
4. Reload format.html in Notepad and create a new bulleted list showing the four seasons. Within each season create a numbered sub list of the appropriate months of the year.
5. Save the file and load it in your Web browser to examine the docum

### 3.4 HTML Comments

Notes may be left in an HTML page to, for example, explain how or why something was done; these notes are often useful for other developers who will be working on the HTML document. These notes, called **comments**, are not displayed by the Web browser, and can only be seen in the HTML source file itself. Web developers make frequent use of HTML comments, and they can found in many Web pages (using the 'View Page Source' menu option).

An HTML comment is given as: `<!-- text in the comment -->` .

#### Activity 7: Comments

In this Activity you will use preformatted text and HTML comments.

1. Load format.html in Notepad.
2. Place an HTML comment before the lists you created.
3. Save the file and load it in a Web browser.

### 3.5 Anchors

To link to another file use the `<A HREF="URL">link</A>` tag.

The term URL is the location of the file to be linked to. It could be on a hard or floppy disk — as in `a:\filename.html` or `c:\my documents\week01\filename.html` — on the same Web server, or on another Web server, as in <http://www.fortunecity.com/username/filename.html>

#### Activity 8: Simple hypertext links

In this Activity you will create four new Web pages: `index.html`, `filetwo.html`, `filethree.html` and `filefour.html`. You will then link them together using relative URLs.

1. Open Notepad and type in the HTML code shown below. (You may find it easier to cut and paste the code from your Web browser into Notepad rather than enter it yourself.)

```
<HTML>
<HEAD>
<TITLE>File name</TITLE>
</HEAD>
<BODY>
<h2>File name</h2>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam
nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam
erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci
tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo
consequat.
<P>
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse
molestie consequat, vel illum dolore eu feugiat nulla facilisis at
vero
eros et accumsan et iusto odio dignissim qui blandit praesent
```

## HTML: Basics

```
luptatum zzzril delenit augue duis dolore te feugait nulla facilisi.  
<P>  
<A HREF="index.html">Homepage</A><BR>  
<A HREF="filetwo.html">Filetwo</A> <BR>  
<A HREF="filethree.html">Filethree</A> <BR>  
<A HREF="filefour">Filefour</A> <BR></BODY>  
</HTML>
```

2. Save this file as **index.htm**. Save the file a further three times using the file names from the list above. Each time also revise file name in the HTML title and body.
3. You should now have four unique files that link to each other. Test these files in your browser by first opening index.html.
4. Now add the following in index.htm to create a hyperlink to the University of Cape Town website.  
<P>  
<A HREF="http://www.uct.ac.za">University of Cape Town</A>
5. Save the file and test it in your browser.

### 3.5.1 Linking to Email Addresses & other Non-Web Links

The previous examples have used the HTTP protocol to inform the browser to load a Web page when you click a hyperlink. Various other protocols may be used. For example, to create a link to an email address use the 'mailto' protocol. Note that this depends on the user's email programme being correctly configured, and so may not always work. However, this feature is commonly used on the Web to contact the webmaster or to get more information from sites.

The following anchor tag creates a mail link:

```
<A HREF="mailto:username@domainname">Email user</A>
```

You can test this by providing your own email address. This was tested and works for a Gmail address.

A subject line for the email message can also be provided:

```
<A HREF="mailto:username@address.com?SUBJECT=e-mail from a friend">user</A>
```

Other protocols that can be used include: ftp://, news://, telnet:// and gopher://. These protocols often require other software besides the Web browser, and, as with emails, if the software is not correctly installed and configured the links will not work.

### 3.5.2 Linking to Sections within Documents

Anchor tags can be used to link to a specific location within an HTML file (even within the same HTML file).

Firstly, a location must be defined using the tag: <A NAME="xxx"></A> with xxx being the location name. A link to the location is created using the tag <A HREF="#xxx">link</A>. If the location is in another document, its file name too must be included as well: <A HREF="URL#xxx">link</A>

#### Activity 9 - Linking to sections within a document

This Activity sets up links to sections within the documents created by Activity 8

1. Open Notepad and load **index.html**.
2. Copy the following text twice into the body of the file above the hyperlinks in order to create a long file. Rename section two to section three when copying it for the second time.

```
section two
```

```
<P>
```

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam  
nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam
```

## HTML: Basics

erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci  
tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo  
consequat..

<P>

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse  
molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero  
eros et accumsan et iusto odio dignissim qui blandit praesent  
luptatum zzril delenit augue dui dolore te feugait nulla facilisi.

<P>

3. Now define the location section two by amending the text in the following way.

```
<A NAME="section_two">section two</A>
```

4. Define the location section three in the same way. Save the file.
5. Save this file as filetwo.html, overwriting the previous file.
6. Re-open index.html and add the following hyperlinks to the top of the <body> section:
  - Section Two
  - Section Three
  - File Two: Section Two
  - File Two: Section Three
7. Ensure that the links work by reloading index.htm in your Web browser.

### 3.5.3 Targeting Windows

In modern Web browsers, anchor tags can specify target windows using the target window attribute.

```
<A HREF="URL" TARGET="New_Window"></A>
```

This specifies where the contents of a selected hyperlink should be displayed. It is commonly used with frames or multiple browser windows, allowing the link to be opened in a specified frame or a new browser window. Windows may have names defined for them, but the underscore should not be used for the first character of any target defined in your documents. Such names are reserved for four special target names:

<code>_blank</code>	The browser always loads a <code>target="_blank"</code> linked document in a newly opened, unnamed window.
<code>_self</code>	This target value is the default for all <code>&lt;A&gt;</code> tags that do not specify a target. It causes the target document to be loaded and displayed in the same frame window as the source document.
<code>_parent</code>	This one is useful for framed sites to create navigation links back to the parent window.
<code>_top</code>	<code>_top</code> forces a break out of a framed site, or to take over the browser window. That means, for example, that if a site has been linked to from within a framed site, clicking on the link only brings up the site inside the frame. The <code>_top</code> target attribute forces the link to take over the entire browser window.

### Activity 10 - Targeting windows

This Activity allows you to try the different target attributes. You will see how they behave by adding them to one or more of the files set up by Activities 9 and 10. For instance, create a link that loads the UCT site in a new browser window.

### 3.5.4 Link Appearance

It is possible to change the colour of a hyperlink to match the colour scheme of the Web page. The underlining of a hyperlink can also be modified or removed.

#### To Do

Read about affecting the appearance of links in your textbooks.

#### Activity 11: Changing the appearance of links

In this Activity you will set up a colour scheme for a Web page by using the **BGCOLOR**, **TEXT**, **LINK**, **VLINK** and **ALINK** attributes of the **BODY** tag.

1. Open Notepad and load **index.html**. Create the following colour schemes for the page using either hexadecimal values or colour names (see your textbooks):
  - a. white background, black text and red hyperlinks.
  - b. black background, white text and cyan hyperlinks.
  - c. your own colour scheme.
2. View the pages in the browser.

## 3.6 Multimedia

### 3.6.1 Graphics

In-line images give the Web much of its visual appeal. They can also cause people using a slow Internet connection to not visit the site; so unless large in-line images are vital, they should either be avoided or alternate pages not using them provided. Small icons, however, have almost negligible transfer cost and can greatly enhance the appearance of your pages.

Use GIF or JPG files for graphics as appropriate:

- Scanned images look better as JPG files. The JPG file format uses variable compression to make the file size smaller, but too much compression causes the picture to lose detail. As a guideline, use a JPG compression level between 60 and 75 percent. JPG is always 24 bit colour or 8 bit grey scale (such as in a black and white photograph).
- GIF is useful for special effects such as transparent artwork (background transparency) and animation. It is 8 bit.

Another format called PNG combines the features JPG and GIF.

The **<IMG>** tag is used to embed an image into a Web document:

```
<IMG SRC="URL" ALT="Alternate Text">
```

The URL is the location of the image file. The ALT attribute specifies text to be displayed if the browser does not display the image.

In-line images are often placed inside anchors. The HTML code to create an image as a hyperlink is:

```
<A HREF="URL"><IMG SRC="URL" ALT="text"></A>
```

When an in-line image is used as a hyperlink, a border is placed around the image. This can be removed using **BORDER=0**. Similarly, a border, of any size, can be placed around a graphic using **BORDER=x**, where **x** is the width of the border in pixels.

## HTML: Basics

Make use of the WIDTH and HEIGHT attributes in the IMG tag to specify the width and height of the image. This allows the browser to layout the page before all the graphics have finished downloading. The width and height can be specified either in pixels or as a percentage:

```
<IMG SRC="URL" WIDTH=x HEIGHT=x> (in pixels)
<IMG SRC="URL" WIDTH=x% HEIGHT=x%> (as a percentage)
```

It is also possible to specify how the graphic is to be placed on the page in relation to the rest of the text. This is done using the ALIGN attribute. By default the bottom of the image is aligned with the bottom of the text — this can be changed using ALIGN=left|right|top|middle|bottom. To include a large graphic that will take a long time to load, first load another (smaller) image while the second larger one loads using LOWSRC="URL" to specify the location of the initial image.

If not specified, the browser expects to find the image in the same folder as the web page. However, it is common to store images in a sub-folder. You must then include the folder name in the src attribute. For example,

```

```

### Activity 12: Graphics

In this Activity you will use the IMG tag and its attributes to create a Web page that includes graphic elements.

1. Open Notepad and begin a new HTML document by entering the main structural tags.
2. Save this file as **image.html**.
3. Now find an image or icon to use in your page. You can save one from any Web page by right-clicking the image and selecting Save As. Save the image in the same directory as **image.html**.
4. This image is going to be embed into the document. Save the file after each instruction and view it in your browser to see the affect.
5. Embed the graphic image in the document using the <IMG SRC="URL"> tag.
6. Include some alternative text to be used <IMG SRC="URL" alt="text">
7. If you can find out the width and height dimensions of the image, include these, otherwise try resizing the image as <IMG SRC="URL" alt="text" width=50 height=50>.
8. Turn the image into a hyperlink by linking it to a page of your choice:  
<A HREF="URL"> <IMG SRC="URL" alt="text" width=50 height=50></A>
9. Turn the border that appears around the image off.  
<IMG SRC="URL" alt="text" width=50 height=50 border=0>.
10. Enter a short paragraph of text.
11. Align the image so that appears on the right side of the Web page.  
<IMG SRC="URL" alt="text" width=50 height=50 border=0 align="right">

### 3.6.2 Objects

This section described how to incorporate other objects objects - such as multimedia and Java applets - into your Web pages:

- The <EMBED SRC="URL"> tag is used to embed media such as background sound. For example,  
<EMBED SRC="music.avi" WIDTH=320 HEIGHT=200 autostart=true loop=3> inserts an AVI movie object into a page, scales it to 320 x 200, starts playing the video, and loops it three times.
- Java applets can be embedded using the <APPLET> tag. For example,  
<APPLET CODE=clock.class  
codebase=http://www.server.com/classes/ height=100  
width=100> </ APPLET> loads a Java applet from a remote Web server.

The <OBJECT> tag is used by multimedia software publishers like Macromedia and Digital Workshop (among others) to embed their programmes into Web browsers. Originally used by Microsoft for its Active-X technology, Netscape has adapted it to some degree. You probably will not be writing code using this tag. Programmes, like Flash and Director, generate the code for you to copy and paste into your HTML files. Some programmes, like Dreamweaver, which is a Web authoring tool, produce sophisticated code snippets based upon the designer's input parameters.

The following is an example that Macromedia Director produced for a Shockwave movie. Notice how the EMBED tag is inside the OBJECT tag.

## HTML: Basics

```
<object classid="clsid:166B1BCA-3F9C-11CF-8075-444553540000"  
codebase="http://download.macromedia.com/pub/shockwave/cabs/director/sw.cab  
#version=7,0,0,0" width="150" height="100">  
  
<param name="src" value="welcome.dcr">  
<embed src="welcome.dcr"  
pluginspage="http://www.macromedia.com/shockwave/download/" width="150"  
height="100">  
  
</embed>  
</object>
```

### 3.6.3 ImageMap

An imagemap is a graphic that has certain areas on it defined as hyperlinks. These areas can do whatever normal HTML links do — email, ftp, gopher, etc. Two very popular uses for image maps are navigation bars and thumbnail sheets.

There are two types of imagemaps: **client-side** imagemaps and **server-side** imagemaps. A **server-side** imagemap requires the imagemap information to be saved within a separate file stored on a server and accessed through a CGI script. This type of imagemap is far more complicated to set up, and is not supported by all servers. Server-side imagemap behaviour varies from system to system, even among different systems using the same server. A server-side imagemap shows mouse co-ordinates at the bottom of the screen.

A **client-side** imagemap (CSIM) requires the imagemap information to be stored within the HTML document. A client-side image map shows the actual URL in the status bar message at the bottom of the browser window. Client-side image maps store the hyperlink information in the HTML document, not in a separate map file as do server-side image maps.

When the user clicks a hotspot in the image, the associated URL is sent directly to the server. This makes client-side image maps faster than server-side image maps because the server does not need to interpret where the user clicked. Client-side image maps are supported by all modern browsers.

## 3.7 Writing Good HTML

This final section is concerned with the production of good HTML, as well as with the importance of testing your Web documents before going 'live'.

There are a number of online validation services that check the validity of a Web page. The World Wide Web Consortium's HTML Validator, for example, checks HTML documents for compliance with W3C HTML Recommendations and other HTML standards.

If there are mistakes in the HTML document, the validator will list the problems — but if it is correct, it provides a message similar to:

```
Valid HTML 4.0  
No errors found!  
Congratulations, this document validates as HTML 4.0 Transitional!
```

Certain HTML editors or authoring tools have built-in HTML checkers or validators; these are sometimes useful, but are not always accurate. Sometimes they notice 'errors' you have intentionally entered, but mostly they can be of great help, particularly when producing complex Web pages. The more sophisticated tools are able to adjust to the level of the target browser, as well as fine tune options such as listing missing ALT attributes inside IMG tags.

### Activity 13: Validating HTML (Optional)

This Activity examines online HTML Validation Services

## HTML: Basics

1. Look at the at W3C's Validator [<http://validator.w3.org>] and submit two or three of your Web pages. What happens?  
Note that you need to give the validators an URL. In other words, your document must exist somewhere on the WWW for it to be validated. You could search for a free hosting service, or place the page on the personal server space most ISPs provide for their clients. Obtaining an URL is not trivial, but it will allow you to validate your code.
2. Try at least one other online validation service and compare the results. Which is the easiest to use? What features of the services do you like or dislike?
3. Discuss your results with the other students in the course in the Discussion Forum.

## 3.8 Discussion and Answers

### 3.8.1 Discussion Topics

At an early stage of learning HTML it is often the little problems that provoke discussion. So you may want to spend some time online discussing the problems you have encountered with your fellow students or tutors.

In particular, you might want to discuss the different format of HTML tags. Some (like bold) have end markers, some (like the line break) do not have end markers, and some (like anchors and images) take arguments. Why do different tags have different formats? What other tags might there be? Is there anything that cannot be done in a tag?

If you managed to validate your code discuss the usability of the Validating Services you investigated as part of Activity 13. In particular, you should:

1. Compare the results obtained from the different online validation services.
2. Discuss their ease of use and their functionality.
3. And finally, you should take an online vote on your preferred online validation service.

### 3.8.2 Discussion of Activity 2

Your HTML document should look something like the one below:

```
<html>
<head>
<title> My first Web page </title>
</head>

<body>
Your name
<i> csc5003 </i>
<b>Internet Commerce</b>

</body>
</html>
```

### 3.8.3 Discussion of Activity 4 and 5

Your solution should look like what is shown in Figure 3.1.

### 3.8.4 Discussion of Activity 6 - Lists

The code to create the lists should look something like the one below. The HTML of particular interest is shown in bold.

```
<UL TYPE=square>
```

## HTML: Basics

```
<LI>Monday</LI>
<LI>Tuesday</LI>
<LI>Wednesday</LI>
<LI>Thursday</LI>
<LI>Friday</LI>
<LI>Saturday</LI>
<LI>Sunday</LI>
<UL>
<OL TYPE=i>
<LI>January</LI>
<LI>February</LI>
<LI>March</LI>
<LI>April</LI>
<LI>May</LI>
<LI>June</LI>
<LI>July</LI>
<LI>August</LI>
<LI>September</LI>
<LI>October</LI>
<LI>November</LI>
<LI>December</LI>
</OL>
<DL>
<DT>Spring
<DD>First season of the year: March - May
<DT>Summer
<DD>Second season of the year: June - August
<DT>Autumn
<DD>Third season of the year: September - November
<DT>Winter
<DD>Fourth season of the year: December - February
</DL>
<UL>
<LI>Spring</LI>
<OL START=3>
<LI>March</LI>
<LI>April</LI>
<LI>May</LI>
</OL>
<LI>Summer</LI>
<OL START=6>
<LI>June</LI>
<LI>July</LI>
<LI>August</LI>
</OL>
<LI>Autumn</LI>
<OL START=9>
<LI>September</LI>
<LI>October</LI>
<LI>November</LI>
</OL>
<LI>Winter</LI>
<OL START=12>
<LI>December</LI>
<LI VALUE=1>January</LI>
<LI>February</LI>
</OL>
</UL>
```

### 3.8.5 Discussion of Activity 7 – Comments

You should include the following tag:

```
<!-- Creating lists -->
```

### 3.8.6 Discussion of Activity 9 – Linking to sections within a document

The link to the specified sections should look as below:

```
<P>  
<A HREF="#section_two">Section two</A><BR>  
<A HREF="#section_three">Section three</A> <BR>  
<A HREF="filetwo.html#section_two">Filetwo-Sec2</A> <BR>  
<A HREF="filetwo.html#section_three">Filetwo-Sec3</A> <BR>
```

### 3.8.7 Discussion of Activity 11 – Changing appearance of links

The code to generate the colour schemes is shown below: white background, black text and red hyperlinks

```
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#FF000000">  
or  
<BODY BGCOLOR="white" TEXT="black" LINK="red">  
<BODY BGCOLOR="#000000" TEXT="#FFFFFF" LINK="#00FFFF">  
or  
<BODY BGCOLOR="black" TEXT="white" LINK="cyan">
```

# Chapter 4. HTML Tables

## Table of Contents

Objectives .....	1
4.1 Introduction to Tables .....	1
4.1.1 What is a Table? .....	1
4.1.2 Why do We Use Tables? .....	2
4.1.3 Creating a Data Table .....	2
4.1.4 Activity 2: HTML Colour Table.....	6
4.2 Using Tables in Page Design .....	6
4.2.1 Using Tables in Page Design .....	6
4.2.2 Flexible Design.....	7
4.2.3 Fixed Design.....	7
4.2.4 Activity 3: Using rowspan .....	8
4.2.5 Activity 4: Using colspan, cellspacing and cellpadding .....	9
4.2.6 Activity 5: More cellspacing and cellpadding .....	12
4.2.7 Activity 6: Fixed and flexible Web page design.....	12
4.2.8 Activity 7: Time Table.....	13
4.3 Review Questions .....	13
4.4 Discussions and Answers.....	13
4.4.1 Correct code for Activity 1 step 1.....	13
4.4.2 Solution to Activity 1 step 5 .....	14
4.4.3 Solution to Activity 2: HTML Color Table .....	14
4.4.4 Discussion Topic .....	15
4.4.5 Answers to Review Questions .....	16

## Objectives

At the end of this unit you will be able to:

- explain why tables are used to organize and display data;
- construct a table using HTML5 tags;
- insert data into the relevant table cells;
- add colour to particular table cells;
- discuss the advantages and disadvantages of fixed and flexible Web page design;
- implement a table in a flexible Web page using relative measurements;
- implement a table in a fixed Web page using pixel measurements.

## 4.1 Introduction to Tables

### 4.1.1 What is a Table?

A table is a grid organized into columns and rows, much like a spreadsheet. An example table is shown below. This table consists of sixteen cells organized into rows and columns. But before beginning to use tables in website design, we should consider the role that they fill. Working with tables in HTML5 has become more powerful due to the new HTML5 table tags and other elements available in HTML5.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

## 4.1.2 Why do We Use Tables?

Tables were initially developed as a method to organize and display data in columns and rows. This chapter discusses such tables. However, tables later became a tool for Web page layout, and as such provide a possible solution for structured navigation. Frames may also be used to provide structured navigation. However, the use of tables over frames is preferred for this purpose, as earlier Web browsers (e.g. Netscape ver.1.0) do not support frames.

### To Do

Read up about tables in your textbooks.

## 4.1.3 Creating a Data Table

Work through Activity 1 in order to understand how tables are created. Bear in mind that rarely is anything achieved which satisfies all of the stated requirements in the first pass. The key to developing perfect Web pages relies on that old adage: "*Learn from your mistakes!*"

Therefore, as long as a start is made, and mistakes are seen as a learning experience, then the design process will eventually succeed.

Please feel free to experiment at any time. If you make mistakes but manage to correct them, take encouragement from this.

### Activity 1: Creating a Table

The objective of this Activity is to create a timetable for CSC5003 students to be displayed on a Web page as shown below:

**CSC503 timetable**

	Monday	Tuesday	Wednesday	Thursday	Friday
6-7pm	look at website	free	Implementation	free	free
7-8pm	take some notes	free	Implementation	free	free

1. Begin a new Web page in your text editor. The header is shown below. When entering the text, try to spot the deliberate mistake and correct it as necessary.

```
<HTML>
  <HEAD>
    <TITLE>
      HTML Table Design
    </HEAD>
  </TITLE>
  <BODY>
  </BODY >
</HTML >
```

The correct code is given at the chapter's end.

2. Save your file as tab\_ex1.html
3. The next stage is to open the table. To open and close a table, use respectively the <TABLE> and </TABLE> tags within the document's BODY.

```

<HTML>
<HEAD>
<TITLE>
  HTML Table
  Design
</HEAD>
</TITLE>
<BODY>
<TABLE>
</TABLE>
</BODY >
</HTML >

```

4. Save the file and load it in your browser. At first you will notice a blank screen as the table is not visible. A border and rows may be added to make the table visible. If you do not specify a border for the table, it will be displayed without borders. When adding a border, its size can be defined in pixels, for example: `<TABLE border=10 style= "width: 80%" >`. Notice the use of the width attribute to set the table to a width of 80% of the screen's size (this can also be defined in pixels). However, it is worth noting that the border attribute is on its way out of the HTML standard! It is better to use CSS by first creating a `<style>` tag within the `<head>` tag then leave using only the style attribute within the table tag. 'td' stands for 'tabular data' and 'th' stands for 'tabular header'.

```

<style>
table, th, td {
  border: 1px
solid black;
}
</style>
....
<TABLE style=
"width: 80%" >

```

5. The `<TR>` tag is used to add rows. Each row is composed of several data cells. Their dimensions can be defined using width and height attributes: `<TD width=25% height=20 bgcolor="darkred">` Notice that the cell's colour can also be defined. Try to create the table below before you look at the solution code under Discussion and Answers at the end of the chapter.

---

red cell	light blue cell
----------	-----------------

**Figure 5.1: Table with one row and two columns**

6. Reopen the file `tab_ex1.html` in your text editor and make the following amendments to `<TABLE>` and `<tr>` tags. Note the `<CENTER>` tag centers the table horizontally and it also centers the text within each cell in the row.

```

<TABLE style= "width: 80%" align = "center">

<tr align = "center">

```

7. Save this file as `tab_ex2.html` and view it in your browser. It should look as below.

red cell	light blue cell
----------	-----------------

**Figure 5.2: Table with centered text**

8. We can see that the text is still not given any specific font. HTML `<FONT>` tag is deprecated in version 4.0, onwards (hence it is not supported in HTML5) and now all fonts are set by using CSS. Try to assign the Comic Sans MS font by making the following addition to the style section. Save the file as `tab_ex4.html`.

```
font-family: Comic Sans MS;
```

This sets all the text in in each cell to have the same font. What if you want to have different fonts in each cell? To do this, you can use the `<p style>` tag within each `<TD>` tag. Modify your `<TD>` tags to the following:

```
<TD width=25% height=70 bgcolor="red"> <p style="font-family: verdana">red  
cell </p> </td>  
<TD width=75% bgcolor="lightblue"> <p style="font-family: Comic Sans MS">  
light blue cell </p></td>
```

9. To add a caption to a table use the `<caption>` tag within the `<table>` body. This caption appears on top of the table. Add the caption “Tabling” to your table thus:

```
<caption> Tabling </caption>
```

10. Save the file as `tab_ex3.html` and view it in your browser. It should look as below.

red cell	light blue cell
----------	-----------------

**Figure 5.3: Table with caption and text with different font**

11. In order to meet the objective of this Activity — that is, to create a timetable for CSC5003 — use the HTML code in the next page. Save this as `tab_ex4.html`. One extra HTML tag needs to be introduced: the `TH` tag, which inserts a table header cell. It is similar to the `TD` tag and has the same attributes (i.e. `align`, `bgcolor`, `height` etc.). However, `TH` is used to set the cell's text apart from the rest of the table's text, usually setting it bold and slightly larger. Now that you have completed Activity 1, you should have a good idea of how to create a basic data table.

```

<HTML>
<HEAD>
<TITLE>
    HTML Table Design
</TITLE>

<style>
table, th, td {
    border: 1px solid black;
}
</style>

</HEAD>
<BODY>

    <TABLE style= "width: 80%" align = "center">
    <caption> CSC503 timetable </caption>
    <tr >
        <td width=50%> </td>
        <th width = 150> Monday </th>
        <th width = 150> Tuesday</th>
        <th width = 150> Wednesday </th>
        <th width = 150> Thursday</th>
        <th width = 150> Friday</th>
    </tr>
    <tr >
        <td > 6-7pm </td>
        <td > Look at website</td>
        <td > free </td>
        <td > Implementation </td>
        <td > free </td>
        <td > free </td>
    </tr>
    <tr >
        <td > 7-8pm </td>
        <td > Take some notes</td>
        <td > free </td>
        <td > Implementation </td>
        <td > free </td>
        <td > free </td>
    </tr>
    </TABLE>

</BODY >
</HTML >

```

Here are instructions on how to organise and display data in a table:

1. Insert the <TABLE> tag and decide on the table's dimensions (if required)
2. Add a row using the <TR> tag
3. In the newly created row, insert a cell <TD> with the necessary dimensions and other attributes
4. Add the data to be displayed
5. Terminate the data cell </TD>
6. Repeat steps 3-5 as necessary
7. Terminate the row </TR>
8. Repeat steps 2-7 until all the necessary rows have been added
9. Terminate the table </TABLE>

## To Do

Look up the basic table structure in your textbooks and on the Internet. Draw up a list of the tags for your own use and reference.

Check your list against this one:

HTML tag	Comments
<TABLE> </TABLE>	Table definition and end tag
<CAPTION> </CAPTION>	Caption definition and end tag
<TR> </TR>	Row definition and end tag
<TD> </TD>	Cell definition and end tag

## 4.1.4 Activity 2: HTML Colour Table

This Activity's objective is to write the HTML code to display the following table. Feel free to add more colours.

Some HTML Colors

Colour	Name	hexidecimal	RGB value
	Salmon	FA8072	250-128-114
	Gold	FFD700	255-215-0

See the end of the chapter for the solution.

### To Do

Read up on 'Spanning Rows and Columns' and 'Table Appearance and Colours' in your textbooks. Add the new tags to your list of table related tags.

## 4.2 Using Tables in Page Design

### 4.2.1 Using Tables in Page Design

Tables are useful for laying out text and images on in Web page. Before continuing with instructions on how to do this, let us first consider why there is a need to manage layout.

It is important to realise that it is not a monitor's absolute size that is usually of interest, but rather its screen **resolution**. While a Web browser can manage to layout a document at any resolution, different resolutions do effect the layout and presentation of an HTML document. Resolution is measured in picture elements, called **pixels**. Typical monitor resolutions are 640x480, 800x600, 1024x870, 1280x1024 and 1600x1200.

Resolution and monitor size are independent of one another: a large monitor can have a low resolution, while a small monitor may have a high resolution. Resolution is determined by the hardware, the user, and the video card driver installed on the computer. A single monitor may have a choice of resolutions.

There is also the issue of a browser's 'live space'. Live space refers to the browser area the Web page is displayed in. This can vary from user to user, as the toolbars and status bars the user chooses to have displayed in the browser will reduce or increase the live space available to a Web page.

It is for all these reasons that the dichotomy between **fixed** and **flexible** Web page design has occurred.

By default, all Web pages are designed with flexibility in mind. Flexibility can be defined as a Web page's ability to resize and adapt to the available resolution, monitor and window sizes. Such an approach has both advantages and disadvantages.

#### Advantages:

- **Default Setting:** therefore no new tags are needed — the Web page fills entire space.
- **Philosophical:** flexibility is the philosophy of the Web i.e. it should be accessible by the greatest number of users.
- **Realistic:** resolutions, monitor and window sizes are always different. Keeping a Web page flexible allows it to be viewed on many available formats.

#### Disadvantages:

- **Uncomfortable:** reading text on large monitors is uncomfortable as the lines are too long.
- **Unpredictability:** the designer often cannot predict how a Web page will appear under varying resolutions and live space sizes.
- **Coherence:** on small monitors, everything may not appear correctly.

## 4.2.2 Flexible Design

While HTML is flexible by default, it should not be confused with thinking a flexible document is disorganised, poorly managed with an unstructured layout. A flexible HTML document can still be structured and organised by using, for instance, tables to create columns of text (as in newspapers), and provide layout design.

Flexible layout can be achieved by using percentage measurements for table dimensions. As an example, view the table below by changing the size of your browser's window (i.e. the live space). Observe that as the window size changes, so does the table size.

The table measurements used in this example are called 'relative' measurements, as the sizes are expressed in terms of a percentage of the screen space.

Books about Computing	
IBM PC Assembly Language and Programming	Abel, P
Object-Oriented Analysis and Design	Booch, G
Demonstration of a Flexible Table.	

## 4.2.3 Fixed Design

Fixed design expresses all dimensions in pixels: the dimensions remain fixed regardless of the size of the device it is viewed on. Such an approach has advantages and disadvantages:

#### Advantages:

- **Consistency:** it is usually important for companies to maintain a consistent image.
- **Control:** fixed design imposes restrictions on line length and hence stops uncomfortably long lines from occurring on Web pages.

#### Disadvantages:

- **Incompatible:** the chosen fixed size may be too large for a user's available live space, causing the user to scroll in order to view the whole page; a fixed Web page may also appear too small, leaving unsightly blank spaces.
- **Totalitarian:** the Web does not want to run the risk of being under too much control, i.e. we do not want every Web page to be identical. Some issues in print design are not transferable to the Web, which has its own idiosyncrasies, giving it the advantage of being independent of the print media.

To develop a fixed Web page using tables, supply all measurements in pixels. The table below is a demonstration

of a fixed table — change the size of your browser window and see the effect it has.

Books about Computing		
Title	Publisher	Author
IBM PC Assembly Language and Programming	Prentice-Hall	Abel, P
Object-Oriented Analysis and Design	Addison-Wesley	Booch, G
Demonstration of a Fixed Table		

### To Do

Read up about standard table templates in your textbooks.

## 4.2.4 Activity 3: Using rowspan

This Activity introduces you to the attribute **rowspan**. The objective of this Activity is to create the following table.

red cell	silver cell
	gold cell

1. Let us start by creating the necessary code for displaying the silver and gold cells.

```
<TABLE style= "width: 30%" align = "center">
<tr >
    <td bgcolor = "silver" height =100> silver cell</td>
</tr>

<tr >
    <td bgcolor = "gold" height =100> gold cell</td>
</tr>
</TABLE>
```

2. Save this file as `adv_tab1.html` and view it in your browser. It should appear as so:

silver cell
gold cell

3. Now we insert a red cell spanning two rows. This is done with the **rowspan** attribute. The following syntax is used when designing tables that include **rowspan**.

`<td rowspan=x>` where **x** is the number of rows to be spanned.

Reopen `adv_tab1.html` and make the amendment shown in bold, below.

```
<TABLE style= "width: 30%" align = "center">
<tr >
  <td bgcolor = "red" rowspan = 2 width = 75> red cell</td>
  <td bgcolor = "silver" height =100> silver cell</td>
</tr>
<tr >
  <td bgcolor = "gold" height =100> gold cell</td>
</tr>
</TABLE>
```

4. Save this exercise as `adv_tab2.html` and view it in your browser. It should now look as below:

red cell	silver cell
	gold cell

## 4.2.5 Activity 4: Using colspan, cellspacing and cellpadding

This Activity introduces you to the attributes **colspan**, **cellspacing** and **cellpadding**. The objective of this Activity is to create the following table.

red cell	
silver cell	gold cell

1. Let us begin by creating the silver and gold cells.

```
<TABLE style= "width: 30%" align = "center">
<tr >
    <td bgcolor = "silver" height =100> silver cell</td>
    <td bgcolor = "gold" height =100> gold cell</td>
</tr>
</TABLE>
```

2. Save this file as adv\_tab3.html and view it in your browser. It should appear as below:



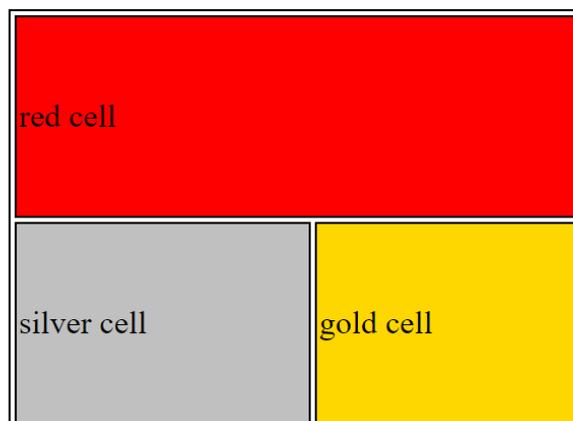
3. For the next step we insert a cell that spans the two columns, using the **colspan** attribute:

`<td colspan=x>` where **x** is the number of columns to be spanned.

Reopen adv\_tab3.html and make the amendment as shown in bold, below:

```
<TABLE style= "width: 30%" align = "center">
<tr>
<td colspan=2 height=100 bgcolor="red">red cell</td>
</tr>
<tr >
    <td bgcolor = "silver" height =100> silver cell</td>
    <td bgcolor = "gold" height =100> gold cell</td>
</tr>
</TABLE>
```

4. Save this exercise as adv\_tab4.html and view it in your browser. It should appear as below.



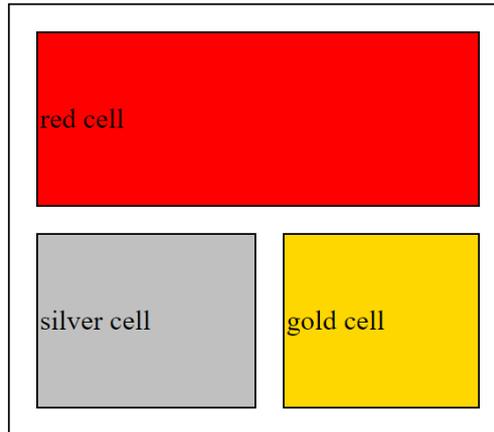
5. The space between the cells is known as the cellspacing. This is controlled with the table attribute **cellspacing**. The following syntax is used with **cellspacing**:

`<table cellspacing=x>` where **x** is the amount of cellspacing required.

Reopen adv\_tab4.html and make the following alterations to the code, as shown in bold.

```
<TABLE style= "width: 30%" align = "center" cellspacing = 15>
<tr>
<td colspan=2 height=100 bgcolor="red">red cell</td>
</tr>
<tr >
<td bgcolor = "silver" height =100> silver cell</td>
<td bgcolor = "gold" height =100> gold cell</td>
</tr>
</TABLE>
```

6. Save this exercise as adv\_tab5.html and view it in your browser. It should appear as below:



7. The space between the text 'red cell' and the cell's border is known as the cellpadding. This can be altered with the attribute **cellpadding**:

`<table cellpadding = x>` where **x** is the thickness, measured in pixels, of the desired cellpadding.

Reopen adv\_tab5.html and make the following amendments to the code, shown in bold.

```
<TABLE style= "width: 30%" align = "center" cellspacing = 2>
<tr>
<td colspan=2 height=100 bgcolor="red">red cell</td>
</tr>
<tr >
<td bgcolor = "silver" height =100> silver cell</td>
<td bgcolor = "gold" height =100> gold cell</td>
</tr>
</TABLE>
```

8. Save this exercise as adv\_tab6.html and view it in your browser. It should appear as required.

red cell	
silver cell	gold cell

## 4.2.6 Activity 5: More cellspacing and cellpadding

The objective of this Activity is to create the table shown below.


This Activity also introduces some of the anomalies that Web page developers deal with during Web page design. In particular, we focus on anomalies with the cellspacing attribute.

Three points should be noted for the above table:

- There are no visible borders, therefore **border=0** is needed.
- There is no cellspacing between cells, so **cellspacing=0**.
- There is no cellpadding, therefore **cellpadding=0**. Now follow these steps to complete the activity.

1. Begin a new file in a text editor and enter the following HTML code:

```
<TABLE style= "width: 30%" height = 30 align = "center" cellspacing = 0
cellpadding = 0>
  <tr>
    <td height=15 bgcolor="darkred"></td>
    <td bgcolor="red"></td>
    <td bgcolor="pink"></td>
  </tr>
  <td height=15 bgcolor="darkblue"></td>
  <td bgcolor="blue"></td>
  <td bgcolor="lightblue"></td>
</TABLE>
```

2. Save your file as adv\_tab7.html and view it in your browser. It should appear as required.

### To Do

Read up about cell spacing in your textbooks. Try to discover common problems and mistakes that Web designers encounter when using tables. While doing this, continue to build on your list of table related tags.

## 4.2.7 Activity 6: Fixed and flexible Web page design

The objective of this Activity is to compare fixed and flexible Web page design.

1. Design a table with two columns, each containing text.
2. First create the table using a flexible page layout. Ensure that your table works as expected by resizing the browser window.
3. Now create the table using a fixed Web page design. Again, check that your answer is correct by resizing the browser window.

## 4.2.8 Activity 7: Time Table

Write the necessary HTML code for your own study timetable. This should look similar to the one shown below. (Hint: use the **colspan** and **rowspan** attributes).

Ian's Timetable

		Morning			lunch	afternoon			evening			
		9-10 am	10-11 am	11-12 am	1-2 pm	2-3 pm	3-4 pm	4-5 pm	5-6 pm	6-7 pm	7-8 pm	8-9 pm
Work	Monday	Development Meeting				Client Meeting			commute	Free		
	Tuesday	in Office				in Office						
Lecture	Wednesday	CSC205				preparation						
	Thursday	MAM200				Tutorials for MAM200			Free			
Research	Friday	Research				Research						

## 4.3 Review Questions

Use the following questions to assess your understanding of HTML tables. Compare your answers to those given below.

1. Initially, why were tables introduced?
2. Why do Web designers prefer to use tables instead of frames?
3. Make a list of possible instructions to develop a table in HTML.
4. Define the difference between fixed and flexible Web page design. State their advantages and disadvantages.
5. You can find the answers at the end of the chapter.

## 4.4 Discussions and Answers

### 4.4.1 Correct code for Activity 1 step 1

The code should appear as below, with the `</TITLE>` tag before the `</HEAD>` tag

```
<HTML>
<HEAD>
<TITLE>
  HTML Table Design
</TITLE>
</HEAD>
```

```
<BODY>
</BODY >
</HTML >
```

## 4.4.2 Solution to Activity 1 step 5

```
<HTML>
<HEAD>
<TITLE>
  HTML Table Design
</TITLE>

<style>
table, th, td {
  border: 1px solid black;
}
</style>

</HEAD>
<BODY>

  <TABLE style= "width: 80%">
  <tr>
    <TD width=25% height=70 bgcolor="red"> red cell </td>
    <TD width=75% bgcolor="lightblue"> light blue cell </td>
  </tr>
  </TABLE>

</BODY >
</HTML >
```

## 4.4.3 Solution to Activity 2: HTML Color Table

Here is the code used to create the table in Activity 2.

```

<HTML>
<HEAD>
<TITLE>
    HTML Table Design
</TITLE>

<style>
table, th, td {
    border: 1px solid black;
}
</style>

</HEAD>
<BODY>

    <TABLE style= "width: 80%" align = "center">
    <caption> Some HTML colors </caption>
    <tr >
        <th width = 150> Color</th>
        <th width = 150> Name</th>
        <th width = 150> Hexadecimal</th>
        <th width = 150> RGB Value</th>
    </tr>
    <tr >
        <td bgcolor = "fa8072"> </td>
        <td > Salmon</td>
        <td > fa8072 </td>
        <td > 250-128-114 </td>
    </tr>
    <tr >
        <td bgcolor = "ffd700"> </td>
        <td > Gold</td>
        <td > ffd700 </td>
        <td > 255-215-0 </td>
    </tr>
    </TABLE>

</BODY >
</HTML >

```

## 4.4.4 Discussion Topic

The purpose of this exercise is to give you the opportunity to discuss the subject of fixed and flexible page design with other students in the course.

Before going to the Discussion Forum, do the following.

1. Go to the O'Reilly's Web site [<http://www.oreilly.com>] and the UCT website [<http://www.uct.ac.za>] and
  - Decide if these pages use fixed or flexible;
  - Consider whether there are any disadvantages with this type of Web page design.
2. Find another example of a Web page that uses a fixed page design, and another that uses a flexible page design.

You are now ready to join the Discussion Forum. In the forum you should be able to:

- Discuss your thoughts on the design principles used to create the O'Reilly and UCT websites
- Share the Web page design examples that you have found and discuss the advantages and disadvantages of fixed and flexible Web page design.

### Additional Resources

You may find these online resources useful in your study of tables.

- Table Tutorial [<http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerPrintable.html#TA>]
- HTML 4.0 Specification Tables Section [<http://www.w3.org/TR/REC-html40/struct/tables.html>]

## 4.4.5 Answers to Review Questions

1. Tables were initially developed as a tool to organise and display data in columns and rows, but they are now also use to support Web page design.
2. Designers prefer to use tables to frames because older versions of browsers do not support frames.
3. List of possible instructions to develop a table in HTML:
  - a. Insert the table tag and decide on its dimensions.
  - b. Add a row with the TR tag.
  - c. Insert a cell in the newly created row, with dimensions and other characteristics.
  - d. Add the data to be displayed.
  - e. Close the data cell.
  - f. Repeat steps three through five as necessary.
  - g. Terminate the row.
  - h. Return to the second step and repeat until all of necessary rows have been added.
  - i. Terminate the table.
4. Fixed Web page design gives designers more control over the Web page, as they can define the exact dimensions of the layout of the page. Flexible Web page design uses relative measurements, and so the element sizes may change, depending on the window size. The advantages of a fixed page design: constant, consistent look to the page; controls line length. Disadvantages of a fixed page design: design may be too large and cause scrolling on the screen. Advantages of a flexible page design: Web page takes up whole page, meets the needs of all resolutions, monitor and window sizes. Disadvantages of a flexible page design: can create unreadable line lengths; possibly unpredictable design.

# Chapter 5. HTML Forms

## Table of Contents

Objectives.....	2
5.1 Introduction.....	2
5.1.1 Processing Forms .....	2
5.2 Creating Forms .....	3
5.2.1 Starting a Form.....	3
5.2.2 Single-line Text Entry .....	4
5.2.3 Multi-line Text Entry.....	6
5.2.4 Radio Buttons .....	7
5.2.5 Check Boxes.....	9
5.2.6 Menu Buttons and Scrolling Lists .....	10
5.2.7 Submit and Reset Buttons .....	13
5.3 HTML5 form elements .....	14
5.3.1 Email address field .....	14
5.3.2 Search.....	14
5.3.3 Url .....	15
5.3.4 Autofocus .....	15
5.3.5 Dates and Times .....	15
5.3.6 Color.....	15
5.3.7 Numbers as Spinboxes .....	15
5.3.8 Numbers as Sliders.....	16
5.4 Using Forms.....	16
5.5 Additional Content and Activities .....	17
5.5.1 Supplementary information on HTML forms.....	17
5.5.2 Additional Activity — Tabular Layout of a Complex Form .....	17
5.6 Review Questions .....	18
5.7 Discussions and Answers.....	20
5.7.1 Discussion of Activity 1 .....	20
5.7.2 Discussion of Activity 2.....	20
5.7.3 Discussion of Activity 3 .....	21
5.7.4 Discussion of Activity 4.....	23
5.7.5 Discussion of Activity 5.....	25
5.7.6 Discussion of Activity 6.....	25
5.7.7 Discussion of Activity 7.....	28
5.7.8 Discussion of Activity 9.....	29
5.7.9 Discussion of Additional Activity .....	31
5.7.10 Answer to Review Question 1 .....	32
5.7.11 Answer to Review Question 2.....	33
5.7.12 Answer to Review Question 3 .....	33
5.7.13 Answer to Review Question 4.....	33
5.7.14 Answer to Review Question 5.....	33
5.7.15 Answer to Review Question 6.....	33
5.7.16 Answer to Review Question 7.....	33
5.7.17 Answer to Review Question 8.....	33
5.7.18 Answer to Exercise 1.....	34
5.7.19 Answer to Exercise 2.....	34
5.7.20 Answer to Exercise 3.....	34

# Objectives

At the end of this chapter you will be able to:

- Create forms with basic elements such as text boxes and buttons;
- Create forms using HTML5 elements such as form validation and email address fields.

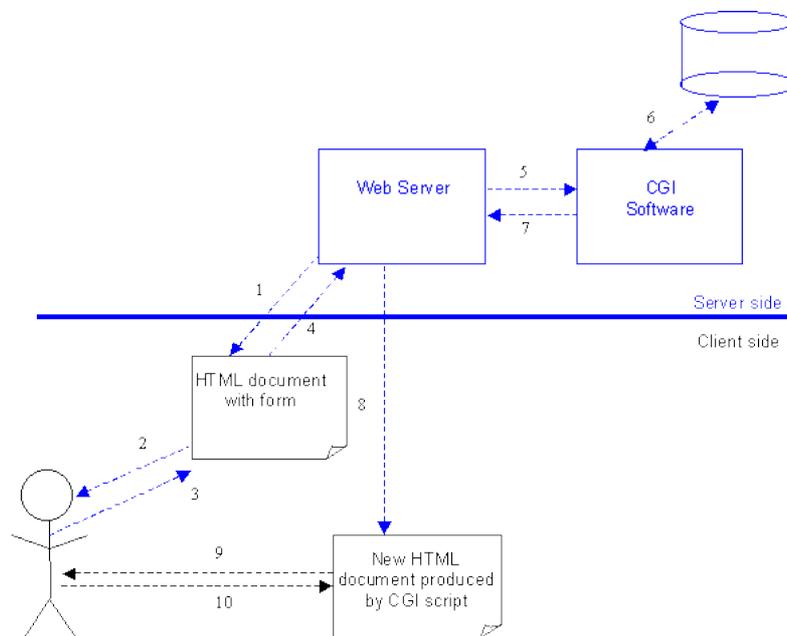
## 5.1 Introduction

Forms are best learnt using a hands on approach. To become proficient with HTML forms you need to create many, sorting out the problematic nuances as you go along. Therefore, the main content of the unit is a series of sections: the first is a short introduction to HTML forms; the second discusses each form element, and involves some textbook study. (You may find it more convenient to postpone activities until you have covered all the form elements).

This introduction covers the main form elements. It also explains the process that occurs when a form is submitted. The main elements of forms are: Text fields; Password fields; Text areas; Radio buttons; Check boxes; Menu buttons and scrolling lists; Submit and reset buttons; and file picker. HTML5 defines a number of new input types that can be used in forms. Examples are Email address fields; web address fields; numbers as spin boxes and sliders; date pickers; search boxes; color pickers; form validation; and required fields. We will look at some of these in this chapter.

### 5.1.1 Processing Forms

Although forms could simply be used to display information, HTML provides them in order to supply a way for the user to interact with a Web server. The most widely used method to process the data submitted through a form is to send it to server-side software typically written in a scripting language, although any programming language can be used. The figure below outlines the kind of processing that takes place.



1. The user retrieves a document containing a form from a Web server.
2. The user reads the Web page and interacts with the form it contains.
3. Submitting the form sends the form data to the server for processing.
4. The Web server passes the data to a CGI programme.
5. The CGI software may use database information or store data in a server-side database.

6. The CGI software may generate a new Web page for the server to return to the user.
7. The user reads the new Web document and may interact with it.

Typically, form data is sent to a server (or to an email address) as a sequence of pairs, each pair being made up of a name and an associated value. The method that this data uses to arrive at its destination depends on the data encoding. Normally the pairs will be sent as binary-encoded characters, making them straightforward to process by software, and easy to read by humans. For example, an on-line store selling used computer parts might use a form when ordering second-hand disk drives; the form would send to the server for processing information identifying the manufacturer, the model name, and maybe quote price thus:

```
manufacturer=syquest&model=e2135&price=45
```

This text represents a sequence of three name/value pairs. The names are **manufacturer**, **model** and **price**, and their associated values are *syquest*, *e2135* and *45*. There is nothing special about the names chosen or the way values are written, except that what is sent depends entirely on what the CGI software expects. If it expected **maker**, **item**, and **cost**, then the data from submitting the form would have to be:

```
maker=syquest&item=e2135&cost=45
```

Quite simply, whatever the processing software expects determines what the HTML form must provide. Often the same person or team develops both form and CGI software, so this is usually of little concern.

Because of the standard way in which the server-side software that process form data is supplied with data, such software is usually referred to as a Common Gateway Interface (CGI) script. Quite often CGI scripts on Unix servers are written in a language called Perl, but languages such as Python are becoming popular; when complex or fast processing is required, C, C++ or Java may be used.

To avoid server side programming when developing forms, and to avoid depending on scripts that may require considerable study, we will mostly use a different method of processing form information: email. In fact, it is very useful to submit form data to an email address, particularly in situations when the data should be seen by a human before being processed by software.

### Review Questions

Do Review Questions 1-2.

## 5.2 Creating Forms

This section explores the main elements found on HTML forms. This is done in a manner matching the way many people develop forms — a little bit at a time. The discussion of each form element involves both reading from your textbook as well as a practical activity. (Some of the activities will take considerable time.) You may prefer to postpone doing Activities 1-7 until you have reached the end of the section on submit and reset buttons.

### 5.2.1 Starting a Form

All forms start with the `<FORM>` tag and end with `</FORM>`. All other form objects go between these two tags.

The form tag has two main properties: **METHOD** and **ACTION**.

**METHOD** refers to **post** or **get**. The **post** attribute will send the information from the form as a text document. The **get** attribute is used mostly with search engines, and will not be discussed. We will generally set **METHOD="post"**.

**ACTION** usually specifies the location of the CGI script that will process the form data. We are not using CGI

## HTML Forms

scripts, and are instead setting this attribute to an imaginary email address (which causes the form data to be emailed to that address).

```
ACTION="mailto:put.your@email.address.here"
```

Putting these together gives us:

```
<FORM METHOD="post" ACTION="mailto:put.your@email.address.here"></FORM>
```

### To Do

Read about Forms in your textbooks.

### Activity 1: Starting a Form

This is the first step in creating a form. You may build on this activity in later ones by using the document you create here as a starting point (or you may prefer to save each different form with a name corresponding to the activity). First, create a new HTML document called form.htm.

Enter the normal <HEAD> and <BODY> tags. Then include the following <FORM> tag:

```
<FORM METHOD="post" ACTION="mailto:put.your@email.address.here">
```

Remember to close the form with the </FORM> tag. Press the return key a few times to move it down the page, as you will be entering further code.

Finally, view your work in a Web browser, but be warned: at the moment there is little to show!

Read Discussion of Activity 1 at the end of the Unit.

## 5.2.2 Single-line Text Entry

A single-line text entry allows the user to input a small amount of text, like a name or an email address.

Please input your name:

This is achieved with the following:

```
Please input your name: <INPUT TYPE="text" SIZE="40" MAXLENGTH="30"  
NAME="personal-name">
```

The tag has the following elements:

<INPUT> is the tag used for most of the form objects.

**TYPE="text"** sets the object to a single-line text field.

**Size="40"** sets the field to show 40 characters.

**MAXLENGTH="30"** means that only a total of 30 characters can be typed in this field.

**NAME="personal-name"** sets the text field's name to be personal-name (this information is part of the form data sent on for further processing). The name is required to identify the value data which will be associated with it. For example, if the text box was for an email address or telephone number, we might set this attribute

## HTML Forms

with a more suggestive value, e.g. `NAME="email"` or `NAME="tel-no"`. The easiest way to choose the name is simply to use the purpose of the field.

**VALUE=...** Another attribute, **VALUE**, can be used to place an initial text in the field. As might be expected, if this text is left unchanged it becomes the value associated with the **NAME** when the form is submitted. Putting an initial value in the field is usually not required, but can be used as a prompt. For example, the following HTML produces the figure that comes after it:

```
Name: <INPUT TYPE="text" NAME="name" SIZE="35"
      VALUE="---please type here---">
```

Name:

Do not confuse the use of 'name' when we refer to the **NAME** attribute of an `<INPUT>` tag with the possibility of using **name** as the text field's actual name, or 'Name' being used in a text field that prompts the user for their own name (as in the example immediately above).

### Exercise 1

After the following HTML form has been rendered by a browser, a user enters their age. The form is subsequently submitted for processing to a CGI script. Write down the name/value pair that is sent to the server-side software. Solution can be found at the end of the chapter.



A browser window showing a form with a text field labeled "Age:" containing the number "39".

### To Do

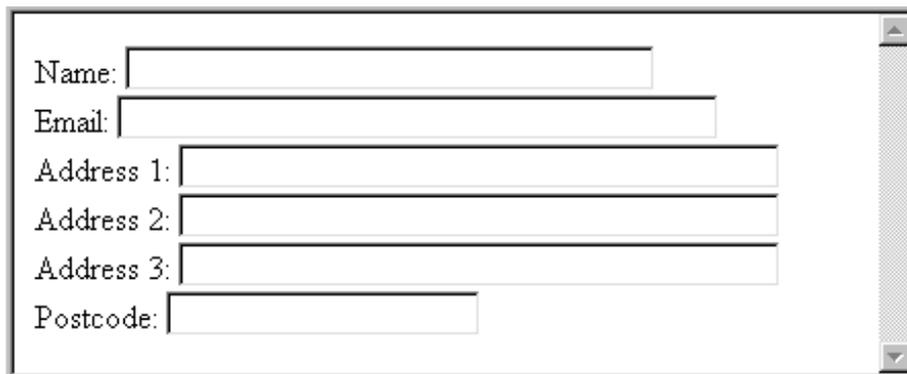
From your textbooks and Internet resources, read about the types of text that you can set for a text-entry, such as password and hidden types. Look at the Additional Content and Activities Section for some on-line resources.

### Activity 2: Single-line Text Entry

In the HTML document you created before (form.htm), or a new one (as you prefer), create

1. A text field for your name that is 35 characters long.
2. A text field for your email that is 40 characters long.
3. Three text fields for your address, each 40 characters long, and an additional smaller field for your postal code.

Test your code to ensure that it produces something as shown below. You may find it convenient to implement and test each part of the form separately.

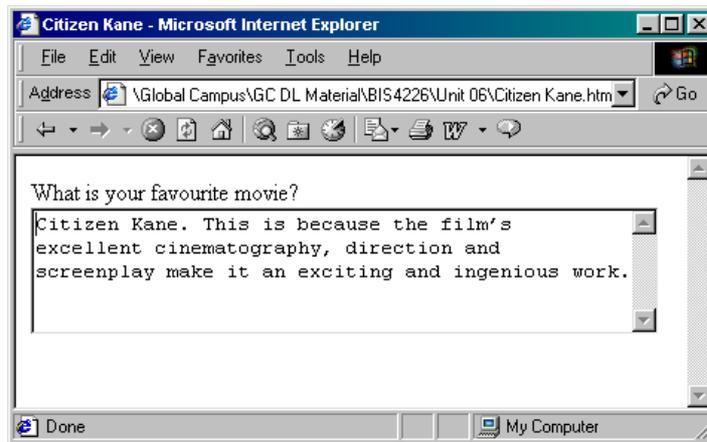


A browser window showing a form with five text fields: Name, Email, Address 1, Address 2, Address 3, and Postcode.

Read Discussion of Activity 2 at the end of the Unit.

## 5.2.3 Multi-line Text Entry

Although it is possible to type as much text as needed into a single line text field, it does become more practical to enter large amounts of text into a multiple-line input field. HTML calls these fields' text areas, as in the example below:



This is achieved with using the `<TEXTAREA></TEXTAREA>` tags. They create a scrollable text area for larger amount of text:

```
<TEXTAREA NAME="movie-comments" COLS="50" ROWS="5"></TEXTAREA>
```

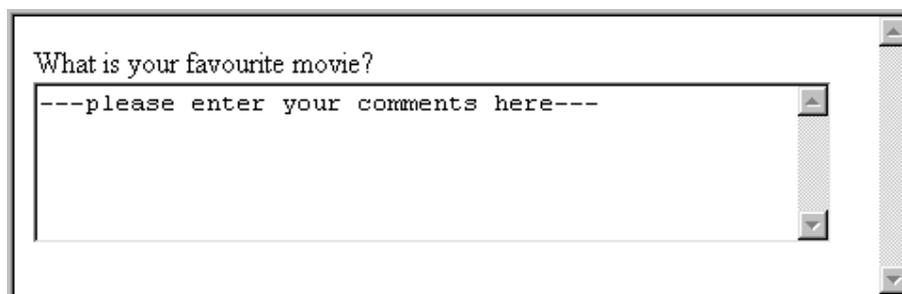
`NAME="movie-comments"` supplies the text field with the given label. Here, because the example allows the user to write about movies, we have named the form element "**movie-comments**".

`COLS="50"` specifies the width, in characters, of the text area. Here 50 characters have been specified.

`ROWS="5"` specifies the height of the text area in rows. This examples specifies five columns. Note that the text that should appear in the multi-line field is placed between the `<TEXTAREA>` and `</TEXTAREA>` tags. So, for example, users could be prompted to input their comments on a movie thus:

```
<TEXTAREA NAME="movie-comments" COLS="50" ROWS="5">
---please enter your comments here--- </TEXTAREA>
```

This would produce the following:



No horizontal scroll bars are present in the above text area examples (just as there rarely are any in a Web browser). This is because, by default, text-wrapping is on. Wrapping is a feature that causes words that cannot fit within a window pane to be moved to the following line or row. Web browsers wrap text (and most other elements) so that when a window is resized, text is redistributed over subsequent lines. A **WRAP** attribute is available, and the default value is `WRAP="ON"`. You can change wrapping to off, which will cause a horizontal scroll bar to appear at the bottom edge of the text area.

### Exercise 2

Change the HTML for the movie opinion text area so that the text does not wrap and a horizontal scroll bar

is provided, as in the figure below:

What is your favourite movie?  
 en Kane. This is because the film's excellent cine:

Solution can be found at the end of the Unit.

## To Do

Read about Text Area in your textbooks.

### Activity 3: Multi-line Text Entry

In an HTML document do the following:

1. Replace the address text fields with one text area large enough to hold the whole address. Use the facilities of the `<INPUT>` and `<TEXTAREA>` tags to prompt the user by including placeholder information in the text fields and text area.
2. Now make the layout of the form more aesthetically pleasing by placing the form in a two-column table, with field labels (e.g. 'Name:', 'Email:') in the left hand column, and the widgets in the right-hand column.
3. Think of three questions relating to the Internet that you might like to ask a user if you were a market researcher trying to gather user information. Create the additional text areas in your form to ask these questions.

Read Discussion of Activity 3 at the end of the chapter.

## 5.2.4 Radio Buttons

Radio buttons are often used on questionnaires to indicate a person's opinion, or their likes and dislikes. They can also be used for 'yes' or 'no' responses. Radio buttons should be used when only one answer from a set of possible answers may be chosen. This is best illustrated by example:

### Example 1

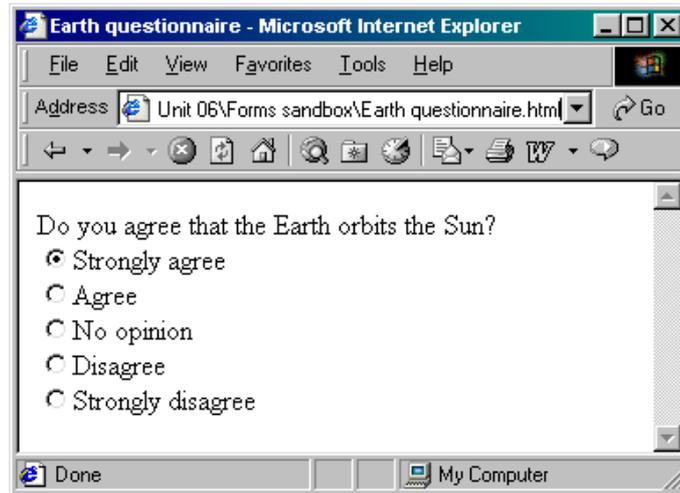
Do you like chocolate?  Yes  No

This is achieved with:

```
Do you like chocolate?
<input type="radio" name="chocolate" value="yes">Yes
<input type="radio" name="chocolate" value="no">No
```

Notice that the *same* name — chocolate — has been used for each element. This is necessary because when the form is submitted only the value of the currently selected radio button will be submitted with the given name. The software processing the data will *either* receive **chocolate=yes** or **chocolate=no**, which allows for straightforward processing.

## Example 2



This is achieved with:

```
Do you agree that the Earth orbits the Sun?<br>
<INPUT TYPE="radio" NAME="orbits" VALUE="strongly_agree">Strongly agree<BR>
<INPUT TYPE="radio" NAME="orbits" VALUE="agree">Agree<BR>
<INPUT TYPE="radio" NAME="orbits" VALUE="no_opinion">No opinion<BR>
<INPUT TYPE="radio" NAME="orbits" VALUE="disagree">Disagree<BR>
<INPUT TYPE="radio" NAME="orbits" VALUE="strongly_disagree">Strongly
disagree<BR>
```

The tag and its attributes are as follows.

**<input>** is the tag used for most of the form objects.

**type="radio"** sets the object to a radio button.

**name="orbits"** labels the entire set of radio buttons. This makes identifying the data easier. For example if the radio button was for the question 'Do you own an automobile?', you might set **name="automobile"**

**VALUE=...** With a set of radio buttons for one question, it is not enough to provide a name only. We need to give each radio button a value so that the form-processing software (CGI script or email) can determine which radio button has been selected. This is where the **value** attribute comes in. Each of the values above is set to the answer for that radio button. This information is sent with the data when the form is submitted. Hence, the form in the above figure would submit **orbits=strongly\_agree**.

**CHECKED** This has not been used in the above example. If it were included, the button is set as if it had been clicked. This is useful if one choice should be made the default.

## To Do

Read about Radio Buttons in your textbooks.

## Activity 4: Radio Buttons

In an HTML document, add these numbered questions and create radio buttons for them.

1. Do you like playing computer games? Yes / No

## HTML Forms

2. How much did you enjoy the 'Star Wars' Trilogy? I enjoyed it / I did not enjoy it / I have not seen it
3. Do you have an email address? Yes / No
4. Chocolate is delicious? strongly agree / agree / neutral / disagree / strongly disagree
5. Then create four more questions of your own choice that use radio buttons.

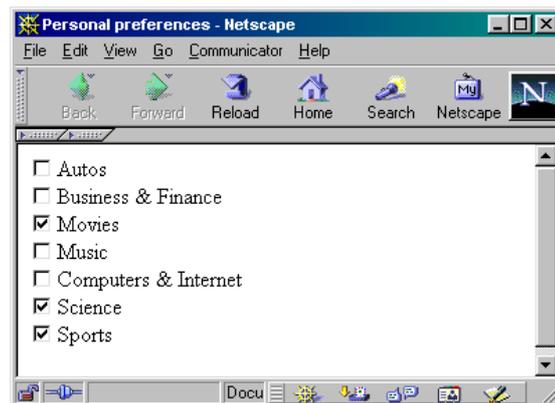
(While working on this activity, you might like to think how the form elements might interact with list structures.)

Read Discussion of Activity 4 at the end of the chapter.

### 5.2.5 Check Boxes

Checkboxes are one of the simplest objects that can be placed on a form. These input elements can only be selected and de-selected. Unlike radio buttons, more than one can be selected at a time.

For example, when signing up for a free e-mail account with GMail [<http://www.gmail.com>] or Hotmail [<http://www.hotmail.com>], a user may well have to fill in a series of forms. One of them is often an interests form, and looks something like this:



```
<INPUT TYPE="checkbox" NAME="autos" VALUE="yes">Autos<BR>
<INPUT TYPE="checkbox" NAME="business"
VALUE="yes">Business & Finance<BR>
<INPUT TYPE="checkbox" NAME="movies"
VALUE="yes">Movies<BR>
<INPUT TYPE="checkbox" NAME="music" VALUE="yes">Music<BR>
<INPUT TYPE="checkbox" NAME="computing"
VALUE="yes">Computers & Internet<BR>
<INPUT TYPE="checkbox" NAME="science"
VALUE="yes">Science<BR>
<INPUT TYPE="checkbox" NAME="sports"
VALUE="yes">Sports<BR>
```

**<INPUT>** is the tag used for most of the form objects. **type="checkbox"** sets the object to a checkbox. **name** is used to supply a name to the checkbox.

**VALUE="yes"** if the item is checked, this is the value that will be associated with the name

## HTML Forms

when the form is submitted for processing. Hence, in the above example, **yes** will be associated with each of the name values **movies**, **science** and **sports**.

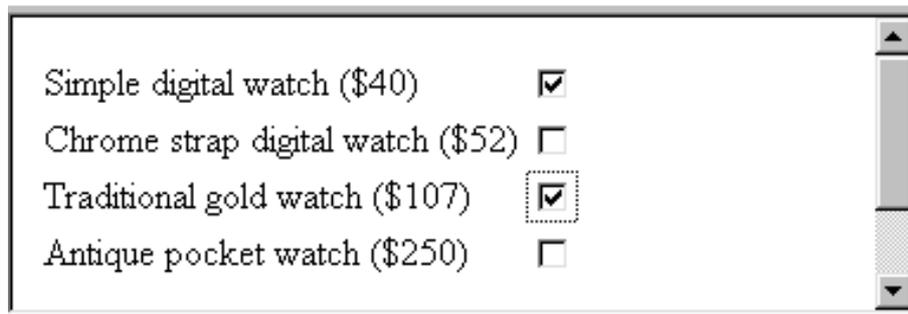
**CHECKED** This has not been used in the above examples. If it were included as an attribute of the tag, the check box would be set as if it had been clicked by the user. This is useful if one or more options are to be offered as defaults.

### To Do

Read about Check Box in your textbooks.

### Activity 5: Check Boxes

Imagine you are developing a website that sells various types of watch. You want to allow customers to select what they want to buy with a set of check boxes, as in the figure below.



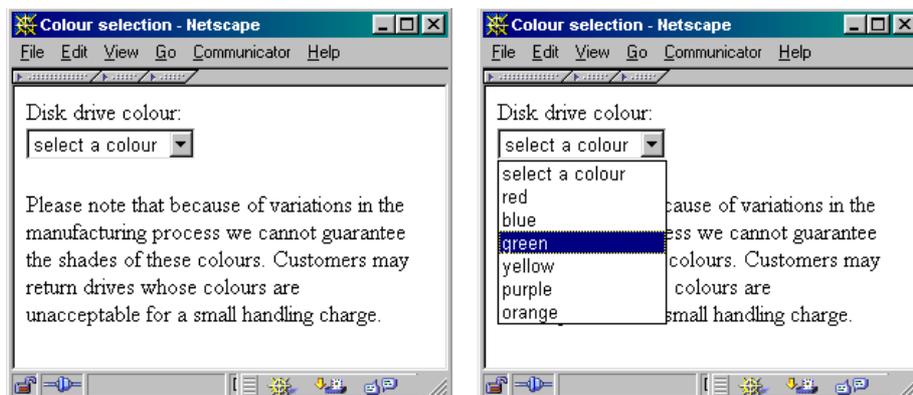
Simple digital watch (\$40)	<input checked="" type="checkbox"/>
Chrome strap digital watch (\$52)	<input type="checkbox"/>
Traditional gold watch (\$107)	<input checked="" type="checkbox"/>
Antique pocket watch (\$250)	<input type="checkbox"/>

Write an HTML form to achieve this (note the tabular layout). Remember that a server-side script may be expecting the prices of the checked items, so these values will need to be transmitted for processing.

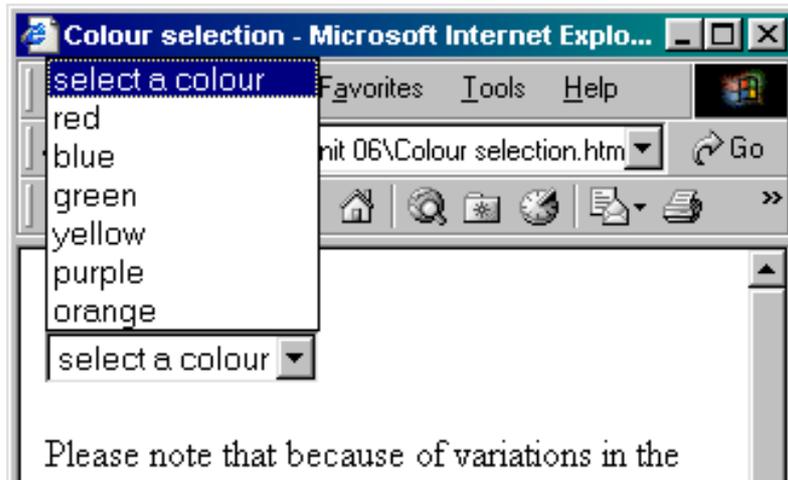
Read Discussion of Activity 5 at the end of the chapter.

## 5.2.6 Menu Buttons and Scrolling Lists

Menu buttons and scrolling lists are useful when you have multiple options to choose from. For example, the following shows a menu button (sometimes imprecisely called a pull-down or drop-down menu). Clicking on the 'select a colour' option on the button displays all the other options in the button's menu. Pulling down to the required option will set the value for this element (for subsequent transmission).



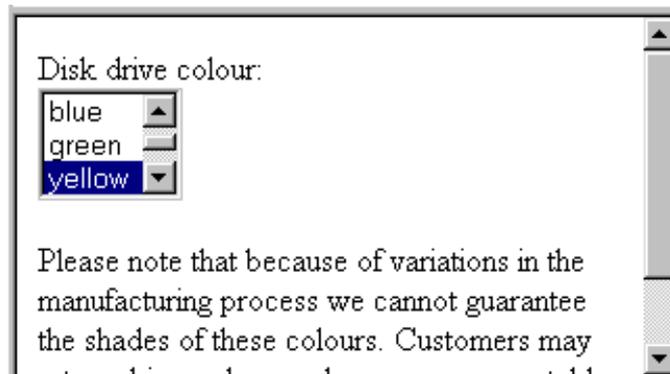
If the menu button is near the bottom of a window, the menu pops up:



Notice that, by convention, the first option of a menu button is usually not an option at all, but a prompt to suggest an action to the user. Hence, in the previous example, 'select a colour' is not an option but a prompt. However, if the form containing this menu button were submitted for processing, the value 'select a colour' would be associated with the named attribute.

Scrolling lists, otherwise known as selection lists, are similar to menu buttons, but they usually display more than one of the available options at a time. They rarely show all options, and the user is required to scroll in order to view them all. If all the options are displayed, no scroll bar is included with the list, and it may not be obvious to the user that they should select an option.

The above example could be redesigned using a scrolling list. The figures below show the three options, and all six options (excluding the prompt):



One clear benefit of these two input elements are that they do not take up as much space as, say, a list of radio buttons. Like radio buttons, they are also used to select one choice one from a set of mutually exclusive options.

The HTML for the menu button and scrolling list (the three-option version) are given below:

## HTML Forms

Disk drive colour:  <SELECT NAME="colour"> <OPTION>select a colour</OPTION> <OPTION>red</OPTION> <OPTION>blue</OPTION> <OPTION>green</OPTION> <OPTION>yellow</OPTION> <OPTION>purple</OPTION> <OPTION>orange</OPTION> </SELECT>	Disk drive colour:  <SELECT NAME="colour" SIZE="3" MULTIPLE> <OPTION>red</OPTION> <OPTION>blue</OPTION> <OPTION>green</OPTION> <OPTION>yellow</OPTION> <OPTION>purple</OPTION> <OPTION>orange</OPTION> </SELECT>
<P>Please note that because of variations in the manufacturing process we cannot guarantee the shades of these colours. Customers may return drives whose colours are unacceptable for a small handling charge.</P>	<P>Please note that because of variations in the manufacturing process we cannot guarantee the shades of these colours. Customers may return drives whose colours are unacceptable for a small handling charge.</P>

<SELECT></SELECT> are used for both menu buttons and scrolling lists (the <INPUT> tag is not used).

**name="colour"** specifies the data name to be transmitted on form submission.

<OPTION></OPTION> are used to specify the option we want to appear in both types of menus.

**size="3"** This attribute sets the number of options that the scrolling list shows. In the above example, the number of options has been set to 3.

**MULTIPLE** This attribute guarantees that the <SELECT> tag results in a scrolling list rather than a menu button. It is not needed if **SIZE** is set to be greater than 1. However, if **MULTIPLE** is omitted and **SIZE=1**, or is omitted, the tag produces a menu button and not a scrolling list.

**SELECTED** This attribute can be included in an <OPTION> tag; it has not been used in the above examples. If it were included, the option is selected by default.

### Exercise 3

A small Egyptian airline flies internally to Egypt while also offering flights to Kuwait, Lebanon, Bahrain, and Oman. Their new website is being developed to promote their business, especially the external services. The website should provide a list of destination options to users. Bearing in mind that most of the company's business is internal and that it wants to advertise its flights to other countries, select an appropriate selection mechanism and arrangement of options. Write the HTML.

A solution can be found at the end of the Unit.

### To Do

Read about creating menus with the <SELECT> tag in your textbooks.

### Activity 6: Menu Buttons and Scrolling Lists

In an HTML document:

1. Create menu buttons for the following:

- Title: Mr / Mrs / Miss/ Ms / Dr
- Age: under 20 (<19) / 20—29 / 30—39 / 40—49 / 50—59 / over 60 (60>)
- Gender: Male / Female
- Status: Employed / Unemployed / Student

2. Create scrolling menus to enter a date of birth using the following:

- Day: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31

## HTML Forms

- Month: January, February, March, April, May, June, July, August, September, October, November, December
- Year: 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1991, 1992, 1993, 1994, 1995, 1996

Read Discussion of Activity 6 at the end of the chapter.

### 5.2.7 Submit and Reset Buttons

Submit and reset buttons have simple uses: they allow the user to send the form data onwards for processing, or to clear the form fields of all entered information.

For example: a user has ordered a large pizza on-line. They have entered four possible topping selections. To send the data for processing, the user clicks on the button labelled 'Order the pizza'. If the user were to make an error and wish to restart, clicking the button labelled 'Clear to restart' will remove the text from the multi-line text area and (as it happens) reset the choice of pizza size to 'Medium'.

We only use fresh ingredients and some pizza toppings may not be available.  
We will supply three from your list.  
Please list preferred toppings in order:

Pizza size:

Small  
 Medium  
 Large

The new elements in this example are the buttons labelled 'Order the pizza' and 'Clear to restart', which are submit and reset buttons respectively. When a submit button is clicked, the Web browser looks at the <form> tag to see how the data should be processed. It will either be sent as an email, as shown below, or sent for processing by a server-side script. Here is HTML for the above example:

```
<FORM METHOD="post" ACTION="mailto:pizza-orders@medpizzas.com.eg">  
<P>We only use fresh ingredients and some pizza toppings  
may not be available. We will supply three from your list.</P> Please  
list preferred toppings in order:<BR>  
<TEXTAREA NAME="toppings" COLS="40" ROWS="5" WRAP=OFF></TEXTAREA><br>  
Pizza size:<BR>  
<INPUT TYPE="radio" NAME="pizza_size" VALUE="small">Small<BR>  
<INPUT TYPE="radio" NAME="pizza_size" VALUE="medium" CHECKED>Medium<BR>  
<INPUT TYPE="radio" NAME="pizza_size" VALUE="large">Large<BR>  
<INPUT TYPE="submit" VALUE="Order the pizza">  
<INPUT TYPE="reset" VALUE="Clear to restart">  
</FORM>
```

Buttons can be customised to perform other functions, such as navigation, by using JavaScript (developers mostly use images and text links for navigation, however) See the Extension section.

A submit button may, in fact, be replaced with an image. For instance, the above example can be

modified as follows:



The `<INPUT>` tag for this submit button uses **image** as the value of the **TYPE** attribute and adds a **SRC** attribute to specify the image to be used, as in:

```
<INPUT TYPE="image" SRC="go-pizza-button.gif">
```

Clicking on an image used as if it were a submit button has exactly the same effect as a standard submit button, in that the form data is dispatched for processing.

Note that there is no corresponding facility to replace a reset button with an image.

## To Do

Read about Submit and Reset Buttons in your textbooks.

### Activity 7: Submit and Reset Buttons

Implement the check box example on personal interests and add submit and reset buttons. Test the reset button by checking several options and then clearing them by clicking Reset. To test the submit button, set the form (using the **ACTION** attribute) to send email to a real address.

Test how the submit button works when using the default encoding (see your textbook). You should receive a binary file attachment at the email address in the **ACTION** attribute. You can open this with a text editor (like Notepad).

Change the `<FORM>` tag to include the encoding **ENCTYPE="text/plain"** and resubmit the form. Examine the email to see how its contents have changed.

Read Discussion of Activity 7 at the end of the Unit.

## 5.3 HTML5 form elements

Create a new HTML file and practice the HTML5 elements in this section.

### 5.3.1 Email address field

The first of these new input types is for email addresses. You can allow the input field to access multiple inputs by using the **multiple** attribute as shown below. For `<input type="email">`: separate each email with a comma. Try typing an email address without the @ field and notice that the browser performs a simple validation.

```
<form>
  <input type="email" multiple>
  <input type="submit" value="Go">
</form>
```

### 5.3.2 Search

Search functions are performed on popular websites such as Google, e-commerce sites as well as personal blogs. It is probably the most common action performed on the Web every day. With HTML5 you can create a simple search action by using:

```
<form >
  <input type="search" name="search">
</form>
```

When you start typing in the search bar you will notice that you may get some suggestions that you can click on, just like you would on Google. Notice also the 'x' on the right which you can use to clear your search results, like you would on Safari browser.

### 5.3.3 Url

The url input type is for web addresses. You can use the multiple attribute to enter more than one URL. Like type="email", a browser will carry out simple validation on these fields and present an error message on form submission. This is likely to include looking for forward slashes, periods, and spaces, and possibly detecting a valid top-level domain (such as .com or .co.uk). Use the url input type like so:

```
<input type="url" name="url">
```

### 5.3.4 Autofocus

If you wish to have one of the field on you form automatically selected when a user loads the webpage, you would give that field the autofocus attribute. Assume that you want to mark the 'url' field from 5.33 with autofocus, you would use the markup:

```
<input type="url" name="url" autofocus>
```

### 5.3.5 Dates and Times

Usually, date pickers are constructed using JavaScript library. HTML5 enables this functionality possible within the browser. Use the markup:

```
<input id="dob" name="dob" type="date">
```

To only show the month and year, use the markup:

```
<input id="expiry" name="expiry" type="month" required>
```

To show the time, use the markup:

```
<input id="time" name="time" type="time">
```

### 5.3.6 Color

The color input type is pretty self-explanatory: it allows the user to select a color and returns the hex value for that color. It is anticipated that users will either be able to type the value or select from a color picker, which will either be native to the operating system or a browser's own implementation. If you are using Chrome you will be able to pick a color from a color picker. To use the color tag, use the markup:

```
<input id="color" name="color" type="color">
```

### 5.3.7 Numbers as Spinboxes

Using HTML5 you can create an input field that accepts minimum and maximum numbers and then you can be able to scroll through the numbers and pick one. Use the following markup:

```
<input type="number" min="0" max="10" step="1" value="0">
```

- type="number" means that this is a number field.

## HTML Forms

- `min="0"` specifies the minimum acceptable value for this field.
- `max="10"` is the maximum acceptable value.
- `step="1"`, combined with the min value, defines the acceptable numbers in the range: 0, 1, 2, and so on, up to the maxvalue.
- `value="0"` is the default value that displays before selecting another number.

### 5.3.8 Numbers as Sliders

By changing the input type from 'number' to 'range' you can change the spinboxes to a slider. Use the markup:

```
<input type="range" min="0" max="10" step="1" value="0">
```

## 5.4 Using Forms

It is time to consider how organizations can make use of forms on their websites. This primarily involves investigating and discussing how commercial enterprises on the Web currently make use of forms.

### Activity 8: How websites use Forms

For this Activity you will need to take a look at a number of websites:

- Amazon Books [<http://www.amazon.com>]
- Loot Free Ads [<http://www.loot.com>]
- Google [<http://www.google.com>]
- Ebay on-line auctions [<http://www.ebay.com>]
- Any of the computer manufacturers (e.g. Dell [<http://www.dell.com>])

Make a note of the following:

- The ways in which websites make use of forms.
- The different types of data that the website are trying to gather. Are there any distinct differences?

To finish this section, you will develop two extensive Web pages that make use of forms.

### Activity 9: Job Application Form

Create an on-line job application form. The application form is for a computer company called SpeedyPC who are advertising for computer programmers. Your form's action should post the application to your email address.

Your application form must have the following elements:

- Position applied for (autofocus), name, nationality, date of birth (selected from an auto picker), address (in a text area), telephone number and email (required).
- Educational history and qualifications.
- Work experience/employment/training in terms of employer history and number of years of experience selected from a slider. Set maximum years of experience to 10 years.
- Personal statement.
- Two referees including names, occupation, relationship, address, telephone.

Read discussion at the end of the chapter.

## 5.5 Additional Content and Activities

### 5.5.1 Supplementary information on HTML forms

Spend 10-20 minutes reviewing the tutorials at one or both of these sites:

- Bare Bones Guide to HTML [<http://werbach.com/barebones/barebone.html>]
- Yale C/AIM Web Style Guide [<http://info.med.yale.edu/caim/manual/contents.html>]

### 5.5.2 Additional Activity — Tabular Layout of a Complex Form

Create a form for processing orders for boxes that looks like those in the figures below. The order form is for a company called 'Land of Boxes'. Their product line consists of only five items:

1. The EconoBox: Cost R5
2. The Standard Box: Cost R10
3. The Premium Box: Cost R15
4. The Deluxe Box: Cost R20
5. The Super Deluxe Box: Cost R30

The order form should include the following elements:

1. Contact details of purchaser, including name, address, telephone number and email.
2. Product details including price, product name, product number and product quantity.
3. Method of payment including credit card type, card number and expiry date.
4. Billing address and delivery address if they are different.
5. The final layout should look as shown in the next two figures. You will need to use tables or CSS styles to achieve this layout.

## HTML Forms

**Land Of Boxes Order Form**

Name:

Email:

Address:

Postcode:

Product	Part Number	Quantity	Unit Price	Subtotal
EconoBox	LB100	<input type="text" value="3"/>	R5	<input type="text" value="R15"/>
Standard Box	LB200	<input type="text" value="4"/>	R10	<input type="text" value="R40"/>
Premium Box/TD>	LB300	<input type="text" value="1"/>	R15	<input type="text" value="R15"/>
Deluxe Box/TD>	LB400	<input type="text" value="0"/>	R20	<input type="text" value=""/>
Super Deluxe Box/TD>	LB500	<input type="text" value="1"/>	R30	<input type="text" value="R30"/>
Total:				<input type="text" value="R100"/>

**Credit Card Details**

Card Type:

Mastercard      Card Number:

Visa      Expiry date:

American Express

Diners Club

Delivery Address:

Read Discussion of Additional Activity at the end of the chapter.

## 5.6 Review Questions

1. Write down the main purpose of HTML forms. Answer at the end of the chapter.
2. What is CGI software?

## HTML Forms

Answer at the end of the chapter.

3. Is the size of the text a user can enter in a text field limited by the value of the **SIZE** attribute in the `<INPUT>` tag?

Answer at the end of the chapter.

4. What attribute must be included in a `<TEXTAREA>` tag to ensure a horizontal scroll bar is included in a multi-line text field?

Answer at the end of the chapter.

5. What is wrong with the following HTML? The code should allow the user to specify their seating arrangement, food, and preferred newspaper, for a flight with a particular airline.

**Flight preferences**

Vegetarian Food  
 European Food  
 Middle Eastern Food  
 Far Eastern Food

London Times  
 Washinton Post  
 Le Figaro

Aisle Seat  
 Window Seat  
 Middle Seat

```
<INPUT TYPE="radio" NAME="flight_prefs" VALUE="veggie">Vegetarian  
Food<BR>  
<INPUT TYPE="radio" NAME="flight_prefs"  
VALUE="euro_food">European Food<BR>  
<INPUT TYPE="radio" NAME="flight_prefs"  
VALUE="midEast_food">Middle Eastern Food<BR>  
<INPUT TYPE="radio" NAME="flight_prefs" VALUE="farEast_food">Far  
Eastern Food<BR><BR>  
<INPUT TYPE="checkbox" NAME="flight_prefs" VALUE="times">London  
Times<BR>  
<INPUT TYPE="checkbox" NAME="flight_prefs" VALUE="post">Washinton  
Post<BR>  
<INPUT TYPE="checkbox" NAME="flight_prefs" VALUE="figaro">Le  
Figaro<BR><BR>  
<INPUT TYPE="radio" NAME="flight_prefs" VALUE="aisle">Aisle  
Seat <BR>  
<INPUT TYPE="radio" NAME="flight_prefs" VALUE="window">Window  
Seat<BR>  
<INPUT TYPE="radio" NAME="flight_prefs" VALUE="middle">Middle  
Seat<BR><BR>
```

Answer at the end of the chapter.

6. How do you set the initial state of a check

box? Answer at the end of the chapter.

7. How do menu buttons and scrolling lists differ in how they save space on a Web

page? Answer at the end of the chapter.

8. Create a scrolling list that shows four items from the following list of personal computer processor types: Motorola 68000, Intel 8088, Intel Pentium MMX, Intel Pentium II, Intel Pentium III, Intel Celeron, PowerPC G3, PowerPC G4, AMD Athlon.

Answer at the end of the chapter

## 5.7 Discussions and Answers

### 5.7.1 Discussion of Activity 1

Your document will probably look something like this:

```
<HEAD>
<TITLE>Form example</TITLE>
</HEAD>
<BODY>
<FORM METHOD="post" ACTION="mailto:put.your@email.address.here">
</FORM>
</BODY>
```

You will notice that nothing shows when this form is rendered because it has none of the interactive elements that make forms useful.

### 5.7.2 Discussion of Activity 2

To produce the text fields one by one:

1. The name text field (below) and its HTML.

Name:

Name: `<INPUT TYPE="text" NAME="name" SIZE="35"><BR>`

2. The email text field (below) and its HTML.

Email:

Email: `<INPUT TYPE="text" NAME="email" SIZE="40"><BR>`

3. The three address fields and the post code field (below), with the HTML.

## HTML Forms

Address 1:

Address 2:

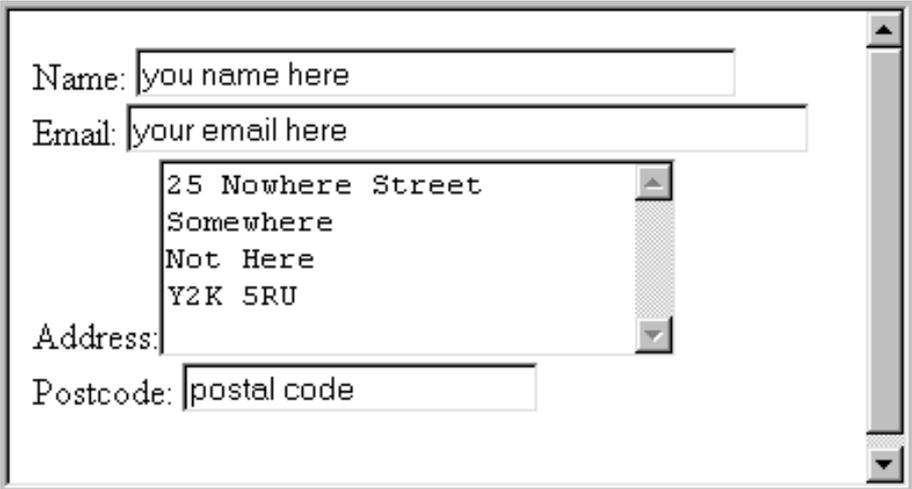
Address 3:

Postcode:

```
Address 1: <INPUT TYPE="text" NAME="address1" SIZE="40"><BR> Address
2: <INPUT TYPE="text" NAME="address2" SIZE="40"><BR> Address 3:
<INPUT TYPE="text" NAME="address3" SIZE="40"><BR> Postcode: <INPUT
TYPE="text" NAME="postcode" SIZE="20"><BR>
```

### 5.7.3 Discussion of Activity 3

1. The multi-line address text area and its HTML are as follows.



The screenshot shows a web form with the following fields and content:

- Name:
- Email:
- Address:
- Postcode:

```
Name: <INPUT TYPE="text" NAME="name" SIZE="35" VALUE="you name here"><BR>
Email: <INPUT TYPE="text" NAME="email" SIZE="40" VALUE="your email
here"><BR>
Address:<TEXTAREA NAME="textfield3" COLS="25" ROWS="5">
25 Nowhere Street Somewhere
Not Here Y2K
5RU
</TEXTAREA><BR>
Postcode: <INPUT TYPE="text" NAME="postcode" SIZE="20" VALUE="postal
code"><BR><BR>
```

2. The nicely laid out version and the table-based HTML follow:

## HTML Forms

Name:	<input type="text" value="you name here"/>
Email:	<input type="text" value="your email here"/>
Address:	<div style="border: 1px solid black; padding: 2px;">25 Nowhere Street Somewhere Not Here Y2K 5RU</div>
Postcode:	<input type="text" value="postal code"/>

```
</FORM>
<TABLE>
<TR>

<TD>Name:</TD>
<TD><INPUT TYPE="text" NAME="name" SIZE="35" VALUE="you name here"></TD>
</TR>
<TR>
<TD>Email:</TD>
<TD><INPUT TYPE="text" NAME="email" SIZE="40" VALUE="your email
here"></TD>
</TR>
<TR>
<TD>Address:</TD>
<TD><TEXTAREA NAME="textfield3" COLS="25" ROWS="5">
25 Nowhere Street
Somewhere
Not Here
Y2K 5RU
</TEXTAREA></TD>
</TR>
<TR>
<TD>Postcode:</TD>
<TD><INPUT TYPE="text" NAME="postcode" SIZE="20" VALUE="postal
code"></TD>
</TR>
</TABLE>
</FORM>
```

Note that the cells have been indented to aid readability. Note that the initial value of the text area (between `<TEXTAREA>` and `</TEXTAREA>` tags) should not be indented unless the text, as rendered by the browser, should itself appear indented.

3. Adding three more text areas to the table structure is more complicated than adding them to a simple (non-table) form: adding a lot of text before the text areas separates them the labels ('Name:', 'Email:', etc.). Aligning the labels to the right brings them closer, as in the figure below. The extra HTML follows the figure:

## HTML Forms

The screenshot shows a web form with the following elements:

- Name:
- Email:
- Address:
- Postcode:
- What is your favourite World Wide Web site and why?
- What is your favourite USENET discussion group and why?
- What it the best designed web site you have come across and why?

```
<TR>
```

```
<TDALIGN="right">What is your favourite World Wide Web site  
and why?</TD>
```

```
<TD><TEXTAREA name="textfield4" cols="40" rows="4">give details  
here</TEXTAREA></TD>
```

```
</TR>
```

```
<TR>
```

```
<TDALIGN="right">What is your favourite USENET discussion group  
and why?</TD>
```

```
<TD><TEXTAREA name="textarea" cols="40" rows="4">give details  
here</TEXTAREA></TD>
```

```
</TR>
```

```
<TR>
```

```
<TDALIGN="right">What it the best designed website you have  
come across and why?</TD>
```

```
<TD><TEXTAREA name="textarea2" cols="40" rows="4">give details  
here</TEXTAREA></TD>
```

```
</TR>
```

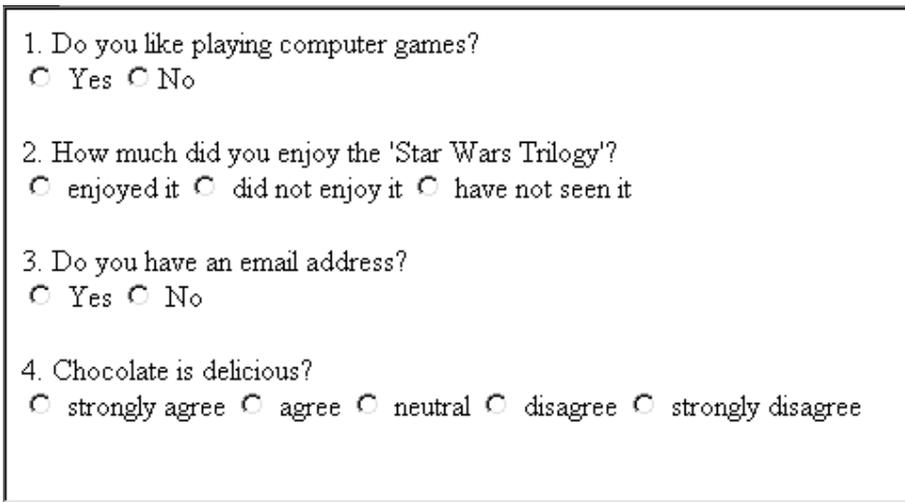
### 5.7.4 Discussion of Activity 4

The following HTML produces the required form (shown after the HTML).

```
<FORM METHOD="post" ACTION=mailto:name@xyz.ac.uk>  
<p>1. Do you like playing computer games? <br>  
<input type="radio" name="games" value="yes"> Yes  
<input type="radio" name="games" value="no">No</p>  
<p>2. How much did you enjoy the 'Star Wars Trilogy'? <br>  
<input type="radio" name="starwars" value="eit"> enjoyed it
```

## HTML Forms

```
<input type="radio" name="starwars" value="dneit"> did not enjoy it
<input type="radio" name="starwars" value="hnsit"> have not seen
it</p>
<p>3. Do you have an email address? <br>
<input type="radio" name="email" value="yes"> Yes
<input type="radio" name="eamil" value="yes"> No</p>
<p>4. Chocolate is delicious? <br>
<input type="radio" name="radiobutton" value="sagree"> strongly
agree
<input type="radio" name="radiobutton" value="agr"> agree
<input type="radio" name="radiobutton" value="neutral"> neutral
<input type="radio" name="radiobutton" value="disagree"> disagree
<input type="radio" name="radiobutton" value="sdisagree"> strongly
disagree</p>
</FORM>
```



1. Do you like playing computer games?  
 Yes  No

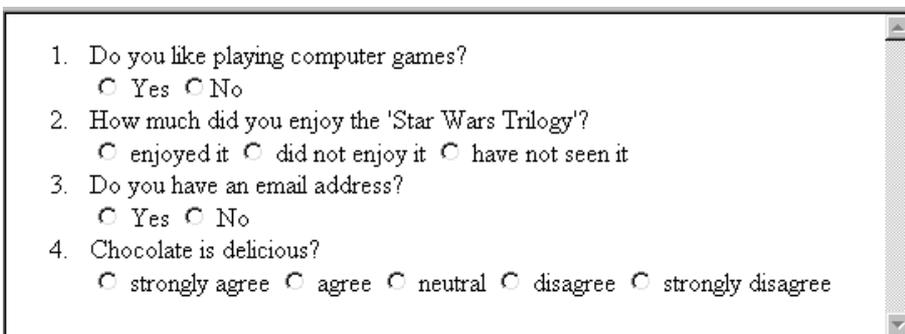
2. How much did you enjoy the 'Star Wars Trilogy'?  
 enjoyed it  did not enjoy it  have not seen it

3. Do you have an email address?  
 Yes  No

4. Chocolate is delicious?  
 strongly agree  agree  neutral  disagree  strongly disagree

Note that a different style of layout can be achieved by combining the form elements with the `<OL>` and `<LI>` tags. Here is a fragment of HTML and its associated layout.

```
<OL>
<li> Do you like playing computer games? <br>
<input type="radio" name="games" value="yes"> Yes
<input type="radio" name="games" value="no">No</li>
...
</OL>
```



1. Do you like playing computer games?  
 Yes  No

2. How much did you enjoy the 'Star Wars Trilogy'?  
 enjoyed it  did not enjoy it  have not seen it

3. Do you have an email address?  
 Yes  No

4. Chocolate is delicious?  
 strongly agree  agree  neutral  disagree  strongly disagree

Here is HTML code for extra questions. Note how the values have been abbreviated. The values are never seen by the user, but are for processing by software, typically a server-side script, and so do not need to be understood by the end users.

```
<p>1. Which of these foods do you prefer? <br>
```

## HTML Forms

```
<input type="radio" name="food" value="italian">Italian
<input type="radio" name="food" value="chinese">Chinese
<input type="radio" name="food" value="indian">Indian </p>
<p>2. Which car do you prefer? <br>
<input type="radio" name="car" value="ferrari">Ferrari
<input type="radio" name="car" value="astin">Aston Martin
<input type="radio" name="car" value="lotus">Lotus
<input type="radio" name="car" value="jaguar">Jaguar</p>
<p>3. Do you like music? <br>
<input type="radio" name="music" value="yes">Yes
<input type="radio" name="music" value="no">No</p>
<p>4. I enjoy reading?<br>

<input type="radio" name="read" value="sa">strongly agree
<input type="radio" name="read" value="a">agree
<input type="radio" name="read" value="n">neutral
<input type="radio" name="read" value="d">disagree
<input type="radio" name="read" value="sd">strongly disagree</p>
```

### 5.7.5 Discussion of Activity 5

Here is the HTML. Note that summing the totals of the selected items requires server-side software. (This is one of the reasons that JavaScript is used to allow such calculations before a form is submitted; see Unit 16.)

```
<TABLE>
<TR>
<TD>Simple digital watch ($40)</TD>
<TD><INPUT TYPE="checkbox" VALUE="40"></TD>
</TR>
<TR>
<TD>Chrome strap digital watch ($52)</TD>
<TD><INPUT TYPE="checkbox" VALUE="52"></TD>
</TR>
<TR>
<TD>Traditional gold watch ($107)</TD>
<TD><INPUT TYPE="checkbox" VALUE="107"></TD>
</TR>
<TR>
<TD>Antique pocket watch ($250)</TD>
<TD><INPUT TYPE="checkbox" VALUE="250"></TD>
</TR>
</TABLE>
```

### 5.7.6 Discussion of Activity 6

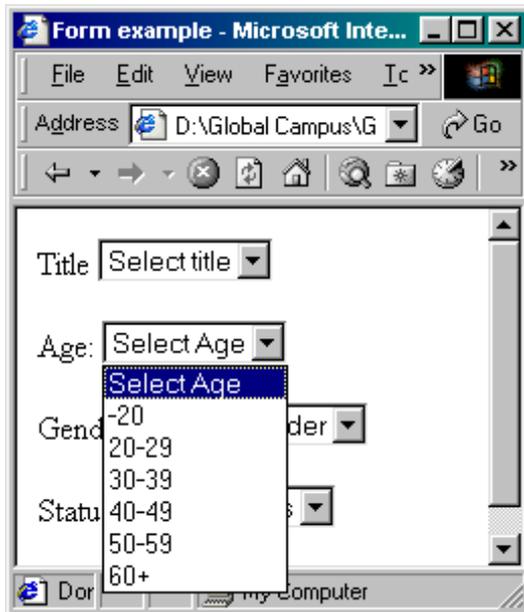
1. The HTML and the output for Title, Age, Gender, employment Status.

```
<FORM METHOD="post" ACTION=mailto:name@xyz.ac.uk>
Title
<SELECT NAME="title">
<OPTION>Select title</OPTION>
<OPTION>Mr</OPTION>
<OPTION>Mrs</OPTION>
<OPTION>Miss</OPTION>
<OPTION>Ms</OPTION>
<OPTION>Dr</OPTION>
```

## HTML Forms

```
</SELECT><BR><BR>
Age :
<SELECT NAME="age">
<OPTION>Select Age</OPTION>
<OPTION>-19</OPTION>
<OPTION>20-29</OPTION>
<OPTION>30-39</OPTION>
<OPTION>40-49</OPTION>
<OPTION>50-59</OPTION>
<OPTION>60+</OPTION>

</SELECT><BR><BR>
Gender :
<SELECT NAME="gender">
<OPTION>Select Gender</OPTION>
<OPTION>Male</OPTION>
<OPTION>Female</OPTION>
</SELECT><BR><BR>
Status :
<SELECT NAME="status">
<OPTION>Select Status</OPTION>
<OPTION>Employed</OPTION>
<OPTION>Unemployed</OPTION>
<OPTION>Student</OPTION>
</SELECT><BR><BR>
</FORM>
```



### 2. The HTML and the output for Title, Age, Gender, employment Status

```
<FORM METHOD="post" ACTION=mailto:name@xyz.ac.uk>
Day :

<SELECT name="occupation" size="3">
<OPTION SELECTED>--Select Day--</OPTION>
<OPTION>1st</OPTION>
<OPTION>2nd</OPTION>
<OPTION>3rd</OPTION>
<OPTION>4th</OPTION>
<OPTION>5th</OPTION>
<OPTION>6th</OPTION>
```

## HTML Forms

```
<OPTION>7th</OPTION>
<OPTION>8th</OPTION>
<OPTION>9th</OPTION>
<OPTION>10th</OPTION>
<OPTION>11th</OPTION>
<OPTION>12th</OPTION>
<OPTION>13th</OPTION>
<OPTION>14th</OPTION>
<OPTION>15th</OPTION>
<OPTION>16th</OPTION>
<OPTION>17th</OPTION>
<OPTION>18th</OPTION>
<OPTION>19th</OPTION>
<OPTION>20th</OPTION>
<OPTION>21st</OPTION>
<OPTION>22nd</OPTION>
<OPTION>23rd</OPTION>
<OPTION>24th</OPTION>
<OPTION>25th</OPTION>
<OPTION>26th</OPTION>
<OPTION>27th</OPTION>
<OPTION>28th</OPTION>
<OPTION>29th</OPTION>
<OPTION>30th</OPTION>
<OPTION>31st</OPTION>
```

```
</SELECT><BR><BR>
```

Month:

```
<SELECT name="select" size="3">
```

```
<OPTION SELECTED>--Select Month---</OPTION>
<OPTION>January</OPTION>
<OPTION>February</OPTION>
<OPTION>March</OPTION>
<OPTION>April</OPTION>
<OPTION>May</OPTION>
<OPTION>June</OPTION>
<OPTION>July</OPTION>
<OPTION>August</OPTION>
<OPTION>September</OPTION>
<OPTION>October</OPTION>
<OPTION>November</OPTION>
<OPTION>December</OPTION>
```

```
</SELECT><BR><BR>
```

Year:

```
<SELECT name="select2" size="3">
```

```
<OPTION SELECTED>--Select Year---</OPTION>
<OPTION>1960</OPTION>
<OPTION>1961</OPTION>
<OPTION>1962</OPTION>
<OPTION>1963</OPTION>
<OPTION>1964</OPTION>
<OPTION>1965</OPTION>
<OPTION>1966</OPTION>
<OPTION>1967</OPTION>
<OPTION>1968</OPTION>
<OPTION>1969</OPTION>
```

```

<OPTION>1970</OPTION>
<OPTION>1971</OPTION>
<OPTION>1972</OPTION>
<OPTION>1973</OPTION>
<OPTION>1974</OPTION>
<OPTION>1975</OPTION>
<OPTION>1976</OPTION>
<OPTION>1977</OPTION>
<OPTION>1978</OPTION>
<OPTION>1979</OPTION>
<OPTION>1980</OPTION>
<OPTION>1982</OPTION>
<OPTION>1983</OPTION>
<OPTION>1984</OPTION>
<OPTION>1985</OPTION>
<OPTION>1986</OPTION>
<OPTION>1987</OPTION>
<OPTION>1988</OPTION>
<OPTION>1989</OPTION>
<OPTION>1990</OPTION>
<OPTION>1991</OPTION>
<OPTION>1992</OPTION>
<OPTION>1993</OPTION>
<OPTION>1994</OPTION>
<OPTION>1995</OPTION>
<OPTION>1996</OPTION>

</SELECT><BR><BR>

</FORM>

```

The image shows a screenshot of a web browser window displaying a form with three dropdown menus. The first menu is labeled 'Day' and has '1st' selected. The second menu is labeled 'Month' and has 'February' selected. The third menu is labeled 'Year' and has '1961' selected. The form is enclosed in a rectangular border with a vertical scrollbar on the right side.

## 5.7.7 Discussion of Activity 7

The HTML code is as follows:

```

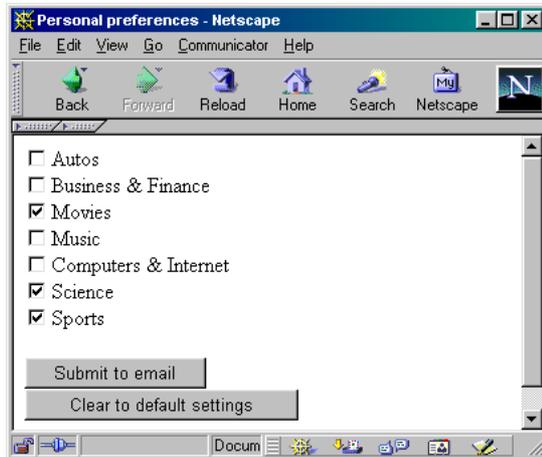
<FORM METHOD="post" ACTION="mailto:your-email@your.isp">
<INPUT TYPE="checkbox" NAME="autos" VALUE="yes">Autos<BR>
<INPUT TYPE="checkbox" NAME="b" VALUE="yes">Business & Finance<BR>
<INPUT TYPE="checkbox" NAME="movies" VALUE="yes">Movies<BR>
<INPUT TYPE="checkbox" NAME="music" VALUE="yes">Music<BR>
<INPUT TYPE="checkbox" NAME="comps" VALUE="yes">Computers & Internet
<BR>
<INPUT TYPE="checkbox" NAME="science" VALUE="yes">Science<BR>

```

## HTML Forms

```
<INPUT TYPE="checkbox" NAME="sports" VALUE="yes">Sports <BR><BR>
<INPUT TYPE="submit" VALUE="Submit to email"><BR>
<INPUT TYPE="reset" VALUE="Clear to default settings">
</FORM>
</BODY>
```

For example, say the boxes labelled 'Movies', 'Science' and 'Sports', were checked, as in the figure below.



With default encoding you would receive an attached file in your email containing:

```
movies=yes&science=yes&sports=yes
```

This can be verified with a text editor.

With **ENCTYPE="text/plain"** you would receive the following in your email:

```
movies=yes
science=yes
sports=yes
```

## 5.7.8 Discussion of Activity 9

Sample HTML and output are given below. Give the form a more presentable layout using either tables or CSS.

```
<!DOCTYPE html>
<html>
<head>
<title>
Job application form
</title>
</head>

<BODY>
  <div class="formLayout">
    <H1>SpeedyPC Job Application Form</H1>
    <HR><BR>
    <FORM METHOD="post" ACTION=mailto:chaombogho@gmail.com>

      <B>Position Applied for:</B> <INPUT type="text" name="position"
size="30" autofocus>

      Job reference code: <INPUT type="text" name="ref" size="15"
bgcolor = "yellow"><BR>

    <HR><BR>
```

## HTML Forms

```
First Name:
<INPUT type="text" name="firstname" size="30"
maxlength="30"><BR> Family Name:
<INPUT type="text" name="familyname" size="30"><BR>

Nationality:
<INPUT type="text" name="nationality" size="30"><BR>

Date of Birth:
<INPUT id="dob" name="dob" type = "date"><BR>

Address:<BR>
<TEXTAREA name="address" cols="30" rows="5"></TEXTAREA>
<BR>

Telephone:
<INPUT type="text" name="phone" size="30" maxlength="30">
Email:
<INPUT type="text" name="email" size="30" maxlength="30"
required /><BR>

<HR><BR>

<B>Educational History</B><BR>

School/College, Course Studied, Qualifications Received, Grades:
<TEXTAREA name="eduhistory" rows="6" cols="70"></TEXTAREA>

<B>Employment History</B><BR>

Employer, Date, Position/responsibilities:
<TEXTAREA name="emphistory" rows="6" cols="70"></TEXTAREA><BR>

<B>Personal Statement:</B><BR>
<TEXTAREA name="statement" rows="6"
cols="70"></TEXTAREA><BR><BR>
<HR><BR>

<B>YEARS OF WORK EXPERIENCE:</B><BR>
<input type="range" min="0" max="10" step="1" value="0">
<HR><BR>

<B>First Referee Details:</B><BR><BR> First name:
<INPUT type="text" name="firstnamel" size="30"> Family name:
<INPUT type="text" name="surnamel" size="30"><BR> Title:
<INPUT type="text" name="title" size="8"> Occupation:
<INPUT type="text" name="occl" size="30"><BR> Relationship:
<INPUT type="text" name="rell" size="20"><BR> Address:<BR>
<TEXTAREA name="address1" rows="4" cols="30"></TEXTAREA><BR>
Telephone:
<INPUT type="text" name="phonel" size="30"> Email:
<INPUT type="text" name="emaill" size="30"><BR><BR>

<B>Second Referee Details:</B><BR><BR> First name:
<INPUT type="text" name="firstnamel" size="30"> Family name:
<INPUT type="text" name="surnamel" size="30"><BR> Title:
<INPUT type="text" name="title" size="8"> Occupation:
<INPUT type="text" name="occl" size="30"><BR> Relationship:
<INPUT type="text" name="rell" size="20"><BR> Address:<BR>
<TEXTAREA name="address1" rows="4" cols="30"></TEXTAREA><BR>
Telephone:
<INPUT type="text" name="phonel" size="30"> Email:
<INPUT type="text" name="emaill" size="30"><BR><BR>
<HR><BR>
<INPUT type="submit" name="Submit" value="Submit">
```

## HTML Forms

```
<INPUT type="reset" name="reset" value="Reset">

</FORM>
</div>
</BODY>

</html>
```

### 5.7.9 Discussion of Additional Activity

Your HTML should look something like this:

```
For example, say you were to check the boxes labelled 'Movies',
'Science' and 'Sports', as in the figure below.
<HEAD>
<TITLE>Land Of Boxes Form</TITLE>
</HEAD>
<H1><FONT SIZE="5" COLOR="coral">Land Of Boxes Order
Form</FONT></H1>
<BODY>
<FORM METHOD="post" ACTION="mailto:name@xyz.co.za"
<TABLE>
<TR>
<TD ALIGN="right">Name:</TD>
<TD><INPUT TYPE="text" NAME="name" SIZE="35"></TD>
</TR>
<TR>
<TD ALIGN="right">Email:</TD>
<TD><INPUT TYPE="text" NAME="email" SIZE="35" VALUE=""></TD>
</TR>
<TR>
<TD ALIGN="right">Address:</TD>
<TD><TEXTAREA NAME="address" COLS="30" ROWS="5"></TEXTAREA></TD>
</TR>
<TR>
<TD ALIGN="right">Postcode:</TD>
<TD><INPUT TYPE="text" NAME="postcode" SIZE="20"></TD>
</TR>
</TABLE><BR><BR>
<TABLE BORDER=1>
<TR>
<TD><B>Product</B></TD>
<TD><B>Part Number</B></TD>
<TD><B>Quantity</B></TD>
<TD><B>Unit Price</B></TD>
<TD><B>Subtotal</B></TD>
</TR>
<TR>
<TD>EconoBox</TD><TD>LB100</TD>
<TD><INPUT type="text" name="Qlb100" size="8"></TD><TD>R5</TD>
<TD><INPUT type="text" name="Sublb100" size="15" value="R"></TD>
</TR>
<TR>
<TD>Standard Box</TD><TD>LB200</TD>
<TD><INPUT type="text" name="Qlb200" size="8"></TD><TD>R10</TD>
<TD><INPUT type="text" name="Sublb200" size="15" value="R"></TD>
</TR>
<TR>
<TD>Premium Box</TD><TD>LB300</TD>
```

## HTML Forms

```
<TD><INPUT type="text" name="Qlb300" size="8"></TD><TD>R15</TD>
<TD><INPUT type="text" name="Sublb300" size="15" value="R"></TD>
</TR>
<TR>
<TD>Deluxe Box</TD><TD>LB400</TD>
<TD><INPUT type="text" name="Qlb400" size="8"></TD><TD>R20</TD>
<TD><INPUT type="text" name="Sublb400" size="15" value="R"></TD>
</TR>
<TR>
<TD>Super Deluxe Box</TD><TD>LB500</TD>
<TD><INPUT type="text" name="Qlb500" size="8"></TD><TD>R30</TD>
<TD><INPUT type="text" name="Sublb500" size="15" value="R"></TD>
</TR>
<TR>
<TD COLSPAN=5 ALIGN="right">
Total: <INPUT type="text" name="total" size="15" value="R"></TD>
</TR>
</TABLE><BR><BR>
<B>Credit Card Details</B><BR>
<TABLE>
<TR>
<TD>Card Type:</TD>
</TR>
<TR>
<TD><INPUT type="radio" name="cardmake"
value="master">Mastercard</TD>
<TD>Card Number:</TD>
<TD><INPUT type="text" name="cardnumber" size="20"></TD>
</TR>

<TR>
<TD><INPUT type="radio" name="cardmake" value="visa">Visa</TD>
<TD>Expiry date:</TD>
<TD><INPUT type="text" name="expiry" size="10" maxlength="5"
value="mm/yy"></TD>
</TR>

<TR>
<TD><INPUT type="radio" name="cardmake" value="amex">American
Express</TD>
</TR>

<TR>

<TD><INPUT type="radio" name="cardmake" value="diners">Diners
Club</TD>
</TR>
</TABLE><BR><BR>

<TABLE>
<TR>
<TD ALIGN="right"><B>Delivery Address:</B></TD>
<TD><TEXTAREA NAME="delivery" COLS="25" ROWS="5"></TEXTAREA></TD>
</TR>
</TABLE><BR><BR>
<INPUT type="submit" name="submit" value="Send order">
<INPUT type="reset" name="reset" value="Clear form">
</FORM>
</BODY>
```

## 5.7.10 Answer to Review Question 1

HTML forms provide a way for the user to interact with a server.

## 5.7.11 Answer to Review Question 2

CGI software is software that adheres to the Common Gateway Interface protocol. A CGI programme is written for a special purpose and performs server-side processing of data submitted from a form.

## 5.7.12 Answer to Review Question 3

No, the **SIZE** attribute in an `<INPUT>` tag merely sets the size of the field that will be rendered by a browser. The

size of text is limited by the **MAXLENGTH** attribute.

## 5.7.13 Answer to Review Question 4

To ensure a horizontal scroll bar is included in a multi-line text field, you must add the `<TEXTAREA>` tag the attribute **WRAP=OFF**

## 5.7.14 Answer to Review Question 5

The problem here is that all the buttons have the name `flight_prefs`, which means that only one name/ value pair will be submitted where three are needed. Further, the first and last sets of radio buttons have unintentionally been made mutually exclusive, again because of the common name

## 5.7.15 Answer to Review Question 6

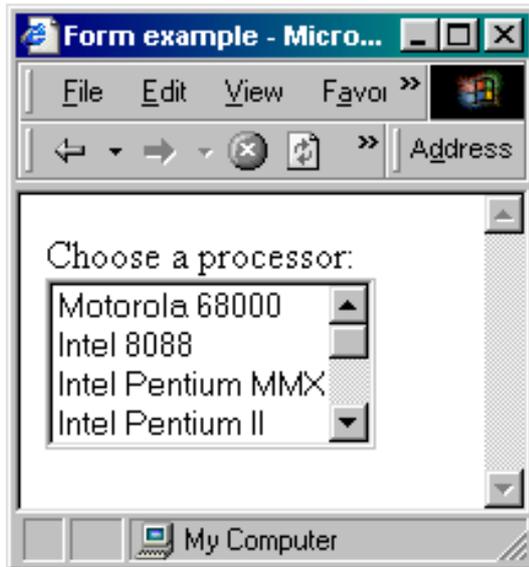
To initially set a check box, include the **CHECKED** attribute in the `<INPUT>` tag.

## 5.7.16 Answer to Review Question 7

Menu buttons require only one line when rendered. The list of options appears over (usually) whatever is below it on the page. Scrolling lists occupy as many lines as specified by the **SIZE** attribute of the `<SELECT>` tag; since all the options are rarely shown, fewer lines than options are usually used.

## 5.7.17 Answer to Review Question 8

Output and HTML are given below:



```
Choose a processor:<BR>
<SELECT name="processor" SIZE=4>
<OPTION>Motorola 68000</OPTION>
<OPTION>Intel 8088</OPTION>
<OPTION>Intel Pentium MMX</OPTION>
<OPTION>Intel Pentium II</OPTION>
<OPTION>Intel Pentium III</OPTION>
<OPTION>Intel Celeron</OPTION>
<OPTION>PowerPC G3</OPTION>
<OPTION>PowerPC G4</OPTION>
<OPTION>AMD Athlon</OPTION></SELECT>
```

## 5.7.18 Answer to Exercise 1

The following name/value pair will be sent to the CGI software. An ampersand (&) would precede or follow it if other pairs were sent.

```
age=39
```

Two layout tips are worth remembering:

1. Use tables to layout the form in an organised way (see Unit 4).
2. You can use the <BR> tag (no closing tag needed) to move text or form objects to the next line. You can use the <P></P> tags to space your work in paragraphs.

## 5.7.19 Answer to Exercise 2

Your HTML code should look something like this:

```
What is your favourite movie?<BR>
<TEXTAREA NAME="movie-comments" COLS="50" ROWS="5"
WRAP=OFF></TEXTAREA>
```

## 5.7.20 Answer to Exercise 3

There may be conflicting requirements here. It is not easy to reflect the fact that the usual destination for customers is within Egypt while promoting the others. For example, a scrolling list with Egypt at the top would appear to satisfy this requirement, but the other countries would not be visible. Breaking alphabetical ordering might distract some users:



```
Destination Country:<BR>
<SELECT NAME="colour" SIZE="3" MULTIPLE>
<OPTION>Egypt</OPTION>
<OPTION>Bahrain</OPTION>
<OPTION>Kuwait</OPTION>
<OPTION>Lebanon</OPTION>
<OPTION>Oman</OPTION>
</SELECT>
```

Alternatively, you could use alphabetical ordering to force the user to scroll through at least the options that precede the most likely destination, Egypt. But there would only be one option before Egypt, so it could be moved to last. To help users find Egypt, it can be pre-selected:



```
Destination Country:<BR>
<SELECT NAME="colour" SIZE="3" MULTIPLE>
<OPTION>Bahrain</OPTION>
<OPTION>Kuwait</OPTION>
<OPTION>Lebanon</OPTION>
<OPTION>Oman</OPTION>
<OPTION SELECTED>Egypt</OPTION>
</SELECT>
```

The final alternative is to use a menu button: it can preserve alphabetic ordering and shows all the other options:

## HTML Forms



Destination Country:

Bahrain

Bahrain

Egypt

Kuwait

Lebanon

Oman

```
Destination Country:<BR>
<SELECT NAME="colour">
<OPTION>Bahrain</OPTION>
<OPTION>Egypt</OPTION>
<OPTION>Kuwait</OPTION>
<OPTION>Lebanon</OPTION>
<OPTION>Oman</OPTION>
</SELECT>
```

Note that, when using a menu button, you should not pre-select Egypt if you want customers for internal flights to see the other options. Pre-selecting does exactly that: it would leave only Egypt visible on the button and most customers would not click on the button to reveal the other options in the menu.

# Chapter 6. HTML5 iFrames

## Table of Contents

Objectives .....	1
6.1 Introduction to i frames.....	1
6.2 i frames Elements.....	1
6.2.1 Web page with one or multiple frames .....	1
6.2.2 Set Width and Height.....	2
6.2.3 Remove the Border .....	2
6.2.4 Use iframe as a Target for a Link.....	2
6.3 Advantages and disadvantages of iframes .....	3

## Objectives

At the end of this chapter you will be able to:

- Understand the use of iframes;
- Create frames using some of the iframe elements.

## 6.1 Introduction to i frames

The basic frames are deprecated in HTML5 and are out of use. The `frameset` tag and its helper tags `frame/noframes` are removed from the modern HTML5 standard. The basic frames had this markup:

```
<FRAMESET COLS="20%,80%">
<FRAME src=left.html>
<FRAME src=right.html>
</FRAMESET>
```

This sets up the frameset to consist of two frames in two columns. The first frame takes 20% of the browser window and the second 80%. A file called `left.html` will appear in one frame and a file called `right.html` in the other. In order to view the framed page in your browser you will need to create these two pages.

HTML5 incorporates the inline frame element, **iframe**, which is a HTML page embedded into the current page.

## 6.2 i frames Elements

### 6.2.1 Web page with one or multiple frames

#### Activity 1: Create a web page with a single Frame

This Activity sets up a webpage with a single frame.

1. Open a text editor, such as Notepad, and enter the `<HEAD>` and `<BODY>` portions of an HTML page.
2. Use the HTML code below to lay out the page with one frame that embeds the UCT website:

```
<iframe src="http://www.uct.ac.za/"></iframe>
```

Save this page as frame1.html and load it in your web browser.

### Activity 2: Create a web page with a multiple frames

This Activity sets up a webpage with two frames.

1. Open a text editor, such as Notepad, and enter the <HTML> and <BODY> portions of an HTML page.

```
<iframe src="right.html"></iframe>
<iframe src="left.html"></iframe>
```

Save this page as frame2.html.

2. Begin another document and enter the following code. Enter the following code:

```
<BODY>This is the left frame</BODY>
```

Save this file as left.html in the same folder as file2.html.

3. Begin another document and enter the following code. Save this as right.html. Note that the bgcolor tag was deprecated in HTML5 and now we use the CSS style.

```
<head>
<style>
BODY
{
    background-color:blue;
}
</style>
</head>

<BODY>
This is the right frame with a blue background
</BODY>
```

4. Now, load frame2.html in your Web browser. You should see that both frames, with the two files loaded in them as appropriate.

## 6.2.2 Set Width and Height

Use the height and width attributes to specify the size.

The attribute values are specified in pixels by default, but they can also be in percent (like "80%"), for example:

```
<iframe src="right.html" width = 200 height = 100></iframe>
```

## 6.2.3 Remove the Border

By default, an iframe has a black border around it.

To remove the border, add the style attribute and use the CSS border property:

```
<iframe src="right.html" style = "border:none"></iframe>
```

## 6.2.4 Use iframe as a Target for a Link

An iframe can be used as the target frame for a link.

### Activity 3: Target iframe

1. In this activity you will create a link that, when clicked, will open the UCT website in a frame.
2. Open frame2.html and make the following changes.

```
<iframe src="right.html" name = "right"></iframe>  
<p><a href="http://www.uct.ac.za/" target="right">Go to UCT</a></p>  
<p>Click on the link above to open UCT website in the right frame</p>
```

3. Save the file and load in in your browser. Click on 'Got to UCT' and the UCT website should load within the right frame.

## 6.3 Advantages and disadvantages of iframes

The major disadvantages of using iframes are:

- Frames can make the production of a website complicated, although current software addresses this problem.
- It is easy to create badly constructed websites using frames. The most common mistake is to include a link that creates duplicate Web pages displayed within a frame.
- Search engines that reference a Web page only give the address of that specific document. This means that search engines might link directly to a page that was intended to be displayed within a frameset.
- Users have become so familiar with normal navigation using tables, the back button, and so on, that navigating through a site that uses frames can be a problem.
- The use of too many frames can put a high workload on the server. A request for, say, ten files, each 1 Kilobyte in size, requires a greater workload than a request for a single 10 Kilobyte file.

The advantages of HTML5 iframes include:

- The main advantage of frames is that it allows the user to view multiple documents within a single Web page.
- It is possible to load pages from different servers in a single frameset.

### To Do

Research iframes and their uses. Discuss the following topics with other students on the forum:

- The merits of designing a Web page with and without iframes.
- The alternatives to using iframes.

Try to cite some examples of good and poor website design using frames of your own.

# Chapter 7. XML

## Table of Contents

Objectives.....	1
7.1 Introduction to Markup Languages.....	2
7.1.1 SGML.....	2
7.1.2 HTML.....	2
7.1.3 XML.....	2
7.1.4 Relationship.....	3
7.2 XML Primer.....	3
7.2.1 Validity and Well-Formedness.....	4
7.2.2 XML Declaration.....	4
7.2.3 Encoding: Unicode.....	4
7.2.4 Document Type Definition (DTD).....	5
7.2.5 Elements / Tags.....	6
7.2.6 Entities.....	7
7.3 Creating your own ML based on XML.....	7
7.4 Parsing and Processing XML.....	8
7.4.1 SAX.....	8
7.4.2 DOM.....	9
7.4.3 SAX vs DOM.....	10
7.5 XML Namespaces.....	10
7.5.1 Default Namespaces.....	11
7.5.2 Explicit Namespaces.....	11
7.6 XML Schema.....	11
7.6.1 Schema Structure.....	11
7.6.2 Sequences.....	11
7.6.3 Nested Elements.....	12
7.6.4 Extensions.....	12
7.6.5 Attributes.....	12
7.6.6 Named Types.....	12
7.6.7 Other Content Models.....	13
7.6.8 Schema Namespaces.....	13
7.6.9 Schema Example.....	13
7.7 Data and Metadata.....	15
7.7.1 Metadata Standards.....	15
7.7.2 Metadata Transformation.....	15
7.8 XPath.....	16
7.8.1 XPath Syntax.....	16
7.9 XSL.....	17
7.10 XSLT.....	17
7.10.1 XSLT Templates.....	17
7.10.2 XSLT Special Tags.....	17
7.10.3 XSLT Language.....	17
7.10.4 XSLT Example.....	18
7.11 Answers.....	19
7.11.1 Answer to Activity 1.....	19

## Objectives

At the end of this chapter you will be able to:

- Explain different markup languages;
- Parse and process XML;
- Use the various XML attributes.

## 7.1 Introduction to Markup Languages

XML (eXtensible Markup Language) is a markup language for documents that contain structured information. *Markup* refers to auxiliary information interspersed with text to indicate structure and semantics. *Documents* does not only refer to traditional text-based documents, but also to a wide variety of other XML *data formats*, including graphics, mathematical equations, financial transaction over a network, and many other classes of information.

Examples of markup languages include LaTeX, which uses markup to specify formatting (e.g. `\emph`), and HTML which uses markup to specify structure (e.g. `<p>`). A markup language specifies the syntax and semantics of the markup tags.

Here is a comparison between plain text and marked up text:

### Plain text:

```
The quick brown
                fox jumped over the lazy dog.
```

### Marked up text:

```
*paragraphstart*The *subjectstart*quick brown fox
                *subjectend* *verbstart*jumped*verbend* over the *objectstart*
                lazy dog*objectend* .*paragraphend*
```

Marked up text aids in semantic understanding, since more information is associated with the sentence than just text itself. This also makes it possible to automatically (i.e. by computer) translate to other formats.

### 7.1.1 SGML

SGML (Standard Generalised Markup Language) specifies a standard format for text markup. All SGML documents follow a Document Type Definition (DTD that specifies the document's structure). Here is an example:

```
<!DOCTYPE uct PUBLIC "-//UCT//DTD SGML//EN">
<title>test SGML document
<author email='pat@cs.uct.ac.za' office=410 lecturer>Pat Pukram
<version>
  <number>1.0
</version>
```

### To do: SGML

Can you see why SGML does not require end tags? Find out more about SGML on the Internet. As a starting point, look at the SGML resources page on the W3 Consortium website [<http://www.w3.org/MarkUp/SGML/>]. Also search for SGML on Google [<http://www.google.com>]

### 7.1.2 HTML

HTML (HyperText Markup Language) specifies standard structures and formatting for linked documents on the World Wide Web. HTML is a subset of SGML. In other words, SGML defines a general framework, while HTML defines semantics for a specific application.

### To Do: HTML

HTML is used to specify both the structure and the formatting of Web documents. Examine the list of HTML tags that you have learnt so far and decide which group each tag belongs into. Read up more on the HTML section of the W3 page [<http://www.w3.org/MarkUp/>].

### 7.1.3 XML

## XML

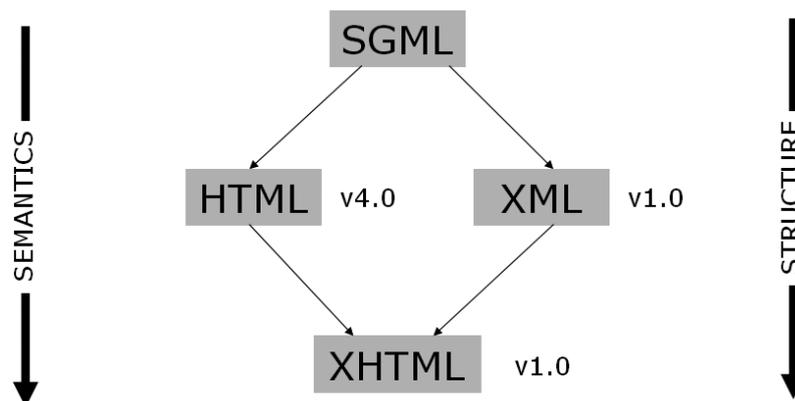
XML, a subset of SGML, was introduced to ease adoption of structured documents on the Web. While SGML has been the standard format for maintaining such documents, its suitability for the Web was poor (for various technical reasons, some of which are discussed later; however, some are beyond the scope of this course). SGML conformity means that XML documents can be read by any SGML system. However, the upside of XML is that XML documents do not require a system capable of understanding the full SGML language.

Both HTML and SGML were considered unsuitable for the use that XML was put to. HTML specifies the semantics of a document (which in HTML's case denotes formatting), but does not provide arbitrary structure. SGML however, does provide arbitrary structure, but is too complex to implement in a Web browser. XML was not designed to replace SGML. As a result, many companies use a SGML to XML filter for their content.

```
<uct>
<title>test XML document</title>
<author email="pat@cs.uct.ac.za" office="410" type="lecturer">Pat
Pukram</author>
<version>
  <number>1.0</number>
</version>
</uct>
```

### 7.1.4 Relationship

The figure below illustrates the relationship between SGML, HTML, XML, and XHTML. XHTML is discussed later in the chapter.



## 7.2 XML Primer

An XML document is a serialised segment of text which follows the XML standard (which can be found at the W3C site [<http://www.w3.org/TR/REC-xml>]).

### To Do: Goals of XML

XML's goals are set out in the W3C recommendations [<http://www.w3.org/TR/REC-xml/#sec-origin-goals>]. Read these recommendations and, if some points are unclear, find out more about them.

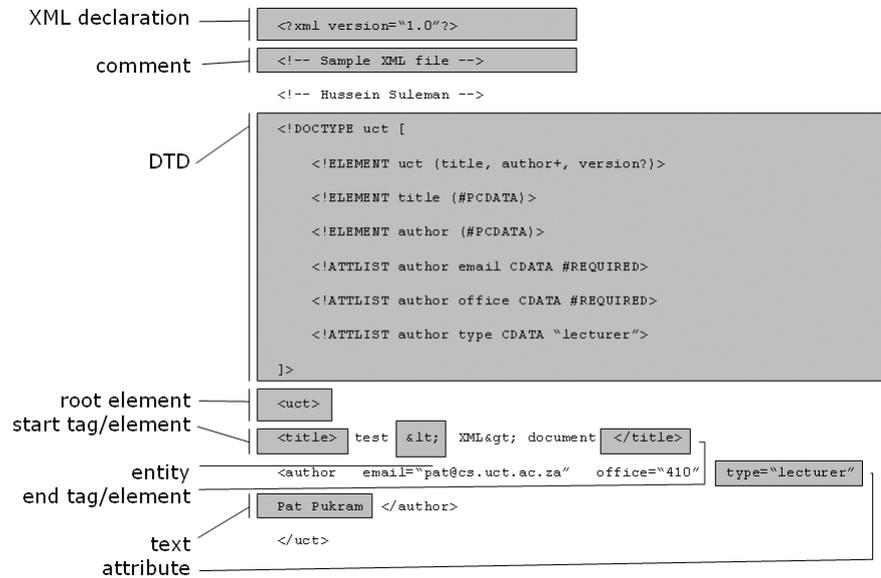
An XML document may contain the following items:

- a XML declaration
- DTDs
- text

## XML

- elements
- processing instructions
- comments
- entity references

The following image contains an example:



### 7.2.1 Validity and Well-Formedness

Well-formed XML documents have a single root element, and start and end tags are properly matched and nested. Valid XML documents strictly follow a DTD (or other formal type definition language). Well-formedness enforces the fundamental XML structure, while validity enforces domain-specific structure. SGML parsers, in contrast, have no concept of well-formedness, so domain-specific structure has to be incorporated into the parsing phase.

#### To Do: Why Validate XML documents

Why do you think it is important to validate XML documents? Discuss this with other students on the online forum.

### 7.2.2 XML Declaration

The XML declaration appears as the first line of an XML document. Its use is optional. An example declaration appears as follows:

```
<?xml encoding="UTF-8" version="1.0" standalone="yes" ?>
```

**encoding** indicates how the individual bits correspond to a character set. See the next section for more detail.

**version** indicates the XML version.

**standalone** indicates whether an external type definitions must be consulted in order to correctly process the document.

### 7.2.3 Encoding: Unicode

The encoding used in the above example, UTF-8, is a Unicode-based encoding scheme. Most XML documents are encoded in the ISO 10646 Universal Character Set (also known as UCS or Unicode). Unicode at first supported 16-bit

## XML

characters, as opposed to ASCII's 8-bits — this 16-bit format could encode 65536 different characters, taken from most of the known languages. This has since been expanded to 32 bits. The simplest encoding mapping this to 4 fixed bytes is called UCS-4. To represent these characters more efficiently, variable length encodings are typically used instead: UTF-8 and UTF-16.

### UTF-16

The Basic Multilingual Plane (characters in the range 0-65535) can be encoded using 16-bit words. Endianness is indicated by a leading Byte Order Mark (BOM) e.g., FF FE = little endian. For more than 16 bits, characters can be encoded using pairs of words and the reserved D800-DFFF range.

```
D800DC00 = Unicode 0x00010000 D800DC01 = Unicode 0x00010001 D801DC01 =  
Unicode 0x00010401 DBFFDFFF = Unicode 0x0010FFFF
```

To match UTF-16 to UCS-4:

```
D801-D7C0 = 0041,  
DC01 & 03FF = 0001  
(0041 << 10) + 0001 = 00010401
```

### UTF-8

UTF-8 is optimal for encoding ASCII text, since the first 128 characters needs only 8 bits to encode. Subsequent characters can be encoded using variable encoding. Here are some examples:

```
Unicode 7-bit = 0vvvvvvv  
Unicode 11-bit = 110vvvvv 10vvvvvv  
Unicode 16-bit = 1110vvvv 10vvvvvv 10vvvvvv  
Unicode 21-bit = 11110vvv 10vvvvvv 10vvvvvv 10vvvvvv etc.
```

Note that the first bits (until the first 0) are used to indicate how many bytes (set of 8 bit) are used to encode the character. Subsequent bytes for the same character encoding begin with 10. The data bits follow each of these header bits (represent by v's in the above examples) in each byte.

To match UTF4 to UTF-8:

```
0001AB45 = 11010 101100 100101  
11110vvv 10vvvvvv 10vvvvvv 10vvvvvv  
= 11110000 10011010 10101100 10100101  
= F09AACA5
```

Note that UTF-8, like UTF-16, is self-segregating to detect code boundaries and prevent errors.

## 7.2.4 Document Type Definition (DTD)

The Document Type Definition (DTD) defines the structure of an XML document. Its use is optional, and it appears either at the top of the document or in an externally referenced location (a file). Here is an example of a DTD:

```
<!DOCTYPE uct [  
  <!ELEMENT uct (title, author+, version?)>  
  <!ELEMENT title (#PCDATA)>  
  <!ELEMENT author (#PCDATA)>  
  <!ATTLIST author email CDATA #REQUIRED>  
  <!ATTLIST author office CDATA #REQUIRED>  
  <!ATTLIST author type CDATA "lecturer">  
  <!ELEMENT version (number)>  
  <!ELEMENT number (#PCDATA)>  

```

## XML

*ELEMENT* defines the basic units of the document's structure. In the above example, they are used to specify different elements (and sub-elements) of documents of type *uct*. The brackets () are used to specify either:

- a list of child elements (sub-element). Each entry in the list can optionally be followed by a symbol each with a different meaning:
- '+': Parent element must have one or more of this child element.
- '\*': Parent element must have zero or more of this child element.
- '?': The existence of child element is optional.

In the example above the *uct* element must consist of a *title* element, at least one *author* element and it can optionally contain a *version* element.

- The data type of the leaf-level element (i.e. elements with no children). In the above example, the *title* element is of type *PCDATA* (text). Alternatively the element could consist of an attribute list, which can be defined using the keyword *ATTLIST* in the following way:

```
<ATTLIST parent_element attribute_name attribute_type (#REQUIRED) ("default_v
```

*#REQUIRED* is optional and can be used to indicate that the attribute is required. *default\_value* is also optional and can be used to specify default value for that attribute. In the above example, the *author* element consists of multiple attributes; namely *email* (required), *office* (required) and *type* (this defaults to "lecturer" if one was not specified).

### Activity 1: DTD

Create a DTD for the following structure:

- **element:** id\_data
- **element:** name
- **element:** firstname
- **element:** middlename (0 or more)
- **element:** lastname
- **element:** date of birth
- **required attribute:** day
- **required attribute:** month
- **required attribute:** year
- **element:** bloodgroup (optional)

You can find the solution at the end of the chapter.

## 7.2.5 Elements / Tags

All elements are delimited by < and >. Element names are case-sensitive and cannot contain spaces (the full character set can be found in the specification). Attributes can be added as space-separated name/value pairs with values enclosed in quotes (either single or double quotes).

```
<sometag attrname="attrvalue">
```

### Structure

## XML

- Elements may contain other elements in addition to text.
- Start tags begin with "<" and end with ">".
- End tags begin with "<" and end with ">".
- Empty tags (i.e. tags with no content, and the start tag is immediately followed by an end tag) can alternatively be represented by a single tag. These empty tags start with "<" and end with ">". In other words, empty tags are shorthand. For example: `<br><br>` is the same as `<br/>`. This means that, when converting HTML to XHTML, all `<br>` tags must be in either of the allowed forms of the empty tags.
- Every start tag must have an end tag and must be properly nested. For example, the following is not well-formed, since it is not properly nested.

```
<x><a>mmm<b>mmm</a>mmm</b></x>
```

The following is well-formed:

```
<x><a>mmm<b>mmm</b></a><b>mmm</b></x>
```

## To Do

Most modern HTML browsers are able to successfully process improperly nested documents. Is this part of the HTML specification? Try to find out more about the similarities and differences between XML and HTML tags.

## Special Attributes

An element tag may indicate additional properties for its contents. For example, `xml:space` is used to indicate if whitespace is significant. In general, it is assumed that all whitespace outside of the tag structure is significant. Another special attributes is `xml:lang` which can be used to indicate the language of the content. For example:

```
<p xml:lang="en">I don't speak</p> Zulu  
<p xml:lang="es">No hablo</p> Zulu
```

## 7.2.6 Entities

Entities begin with '&' and end with ';' . Entities represent (refer to) previously defined textual content, usually defined in a DTD. For example, `&copy;` can only be used if the `ISOLat1` entity list is included. Character entities can be used to refer to Unicode characters. For example, `&#23;` refers to decimal character number 23 and `&#x0041;` refers to hex character number 41. Entities can also refer to predefined escape sequence entities such as `&lt;` (<), `&gt;` (>), `&apos;` ('), `&quot;` (") and `&amp;` (&).

## 7.3 Creating your own ML based on XML

Relational data, such as the set below, can be encoded into XML format as follows:

class	CS&614	
students	3	
marks	vusi	12

## XML

john	24
nithia	36

This data could be encoded as shown below, although there are many other ways of doing this. Can you rewrite it differently?

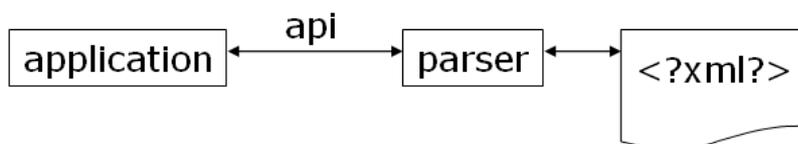
```
<class>CS&614</class>
<students>3</students>
<mark>
  <student_name>vusi</studentname>
  <mark_obtained>12</mark_obtained>
</mark>
<mark>
  <student_name>john</student_name>
  <mark_obtained>24</mark_obtained>
</mark>
<mark>
  <student_name>nithia</student_name>
  <mark_obtained>36</mark_obtained>
</mark>
```

Now encode the following in XML:

Name	Age	Occupation
Tsepo	20	Student
James	19	Waiter
Molly	27	Executive

## 7.4 Parsing and Processing XML

XML parsers process both the data contained in an XML document, as well as the data's structure. In other words, they expose both to an application, as opposed to regular file input where an application only receives content. Applications manipulate XML documents using APIs exposed by parsers. The following diagram show the relationship.



Two popular APIs are the Simple API for XML (SAX) and Document Object Model (DOM).

### 7.4.1 SAX

The Simple API for XML (SAX) is an event-based API that uses callback routines or event handlers to process different parts of an XML documents. To use SAX, one needs to register handlers for different events and then parse the document. Textual data, tag names and attributes are passed as parameters to the event handlers.

#### To Do

Read up about SAX on the Internet. A good page to start is the SAX project home page [<http://sax.sourceforge.net/>].

Using handlers to output the content of each node, the following output can be trivially generated:

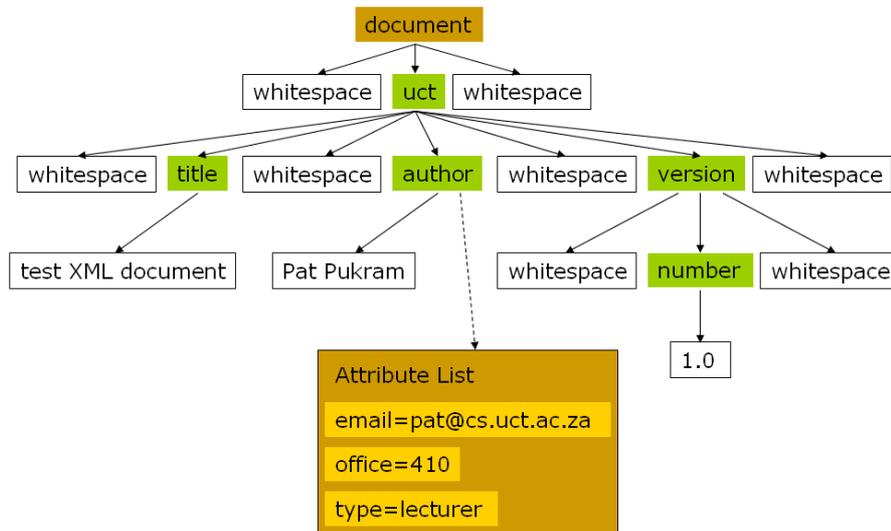
## XML

```
start document start tag : uct start tag : title
content : test XML document end tag : title
start tag : author content : Pat Pukram end tag : author start tag :
version start tag : number content : 1.0
end tag : number end tag : version end tag : uct
end document
```

### 7.4.2 DOM

The Document Object Model (DOM) defines a standard interface to access specific parts of the XML document, based on a tree-structured model of the data. Each node of the XML document is considered to be an object with methods that may be invoked to get/set its contents/structure, or to navigate through the tree. DOM v1 and v2 are W3C [www.w3c.org] standards with DOM3 having become a standard as of April 2004.

Here is a DOM tree of our example:



You might be able to understand the following code. Perl is a popular language to use in DOM processing because of its text-processing capabilities. Java is also popular because of its many libraries and Servlet support.

#### Step-by-step Parsing

```
# create instance of parser my $parser = new DOMParser;
# parse document
my $document = $parser->parsefile ('uct.xml');
# get node of root tag
my $root = $document->getDocumentElement;
# get list of title elements
my $title = $document->getElementsByTagName ('title');
# get first item in list
my $firsttitle = $title->item(0);
# get first child - text content
my $text = $firsttitle->getFirstChild;
# print actual text print $text->getData;
```

#### Quick-and-dirty Approach

```
my $parser = new DOMParser;
my $document = $parser->parsefile ('uct.xml');
print $document->getDocumentElement->getElementsByTagName ('title')->item(0)->g
```

## DOM Interface Subset

Different level of the DOM tree have different attributes and methods:

- **Document**
- **attributes:** documentElement
- **methods:** createElement, createTextNode, ...
- **Node**
- **attributes:** nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes
- **methods:** insertBefore, replaceChild, appendChild, hasChildNodes
- **Element**
- **methods:** getAttribute, setAttribute, getElementsByTagName
- **NodeList**
- **attributes:** length
- **methods:** item
- **CharacterData**
- **attributes:** data

### DOM Bindings

The DOM has different bindings in different languages. Each binding must cater for how the document is parsed — this is not part of DOM. In general, method names and parameters are consistent across bindings. Some bindings define extensions to the DOM, for example, to serialise (turn into a linear data structure) an XML tree.

### To Do

Read up more about DOM on the Internet. You can start by looking at the W3C's section on DOM [<http://www.w3.org/DOM/>]. Also make use of search engines.

## 7.4.3 SAX vs DOM

Below we compare SAX and DOM.

- DOM is a W3C standard while SAX is a community-based "standard".
- DOM is defined in terms of a language-independent interface, while SAX is specified for each implementation language (with Java being the reference).
- DOM requires reading the whole document to create an internal tree structure while SAX can process data as it is parsed. In general, DOM uses more memory to provide random access.

## 7.5 XML Namespaces

Namespaces partition XML elements into well-defined subsets in order to prevent name clashes between elements. If two XML DTDs define the tag "title", which one is implied when the tag is taken out of its document context (e.g., during parsing)? Namespaces disambiguate the intended semantics of XML elements.

## 7.5.1 Default Namespaces

If no namespace is specified for an element, it is placed in the default namespace. An element's namespace (and the namespace of all of its children) is defined with the special "*xmlns*" attribute on an element. Example:

```
<uct xmlns="http://www.uct.ac.za">
```

Namespaces are specified using URIs, thus maintaining uniqueness. Universal Resource Locator (URL) = location-specific

Universal Resource Name (URN) = location-independent Universal Resource Identifier (URI) = generic identifier

## 7.5.2 Explicit Namespaces

Multiple active namespaces can be defined using prefixes. Each namespace is declared with the attribute "*xmlns:ns*", where *ns* is the prefix to be associated with the namespace. The containing element and its children may then use this prefix to specify their membership to a namespace other than the default.

```
<uct xmlns="http://www.uct.ac.za" xmlns:dc="http://somedcns">
  <dc:title>test XML document</dc:title>
</uct>
```

## 7.6 XML Schema

A XML Schema is an alternative to the DTD for specifying an XML document's structure and data types. It is capable of expressing everything a DTD can, and more. Similar, alternative languages exist, such as RELAX and Schematron, but XML Schemas are a W3C standard.

### 7.6.1 Schema Structure

Elements are defined using `<element name="..." type="..." minOccurs="..." maxOccurs="...">`, where:

- *name* refers to the tag.
- *type* can be custom-defined or one of the standard types. Common predefined types include *string*, *integer* and *anyURI*.
- *minOccurs* and *maxOccurs* specify how many occurrences of the element may appear in an XML document. *unbounded* is used to specify no upper limits.

Example: `<element name="title" type="string" minOccurs="1" maxOccurs="1"/>`

### 7.6.2 Sequences

Sequences of elements are defined using a `complexType` container:

```
<complexType>
  <sequence>
    <element name="title" type="string"/>
    <element name="author" type="string" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

Note: Defaults for both `minOccurs` and `maxOccurs` are 1.

## 7.6.3 Nested Elements

Instead of specifying an atomic type for an element, its type can be elaborated as a structure. This corresponds to nested XML elements.

```
<element name="uct">
  <complexType>
    <sequence>
      <element name="title" type="string"/>
      <element name="author" type="string" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
```

## 7.6.4 Extensions

Extensions are used to place additional restrictions on an element's content. For instance, the content can be restricted to be a value from a given set:

```
<element name="version">
  <simpleType>
    <restriction base="string">\
      <enumeration value="1.0"/>
      <enumeration value="2.0"/>
    </restriction>
  </simpleType>
</element>
```

The content can be forced to conform to a regular expression:

```
<element name="version">
  <simpleType>
    <restriction base="string">
      <pattern value="[1-9]\.[0-9]+"

```

## 7.6.5 Attributes

Attributes can be defined as part of complexType declarations.

```
<element name="author">
  <complexType>
    <simpleContent>
      <extension base="string">
        <attribute name="email" type="string" use="required"/>
        <attribute name="office" type="integer" use="required"/>
        <attribute name="type" type="string"/>
      </extension>
    </simpleContent>
  </complexType>
</element>
```

## 7.6.6 Named Types

Types can be named and referred to at the top level of the XSD.

```
<element name="author" type="uct:authorType"/>
```

## XML

```
<complexType name="authorType">
  <simpleContent>
    <extension base="string">
      <attribute name="email" type="string" use="required"/>
      <attribute name="office" type="integer" use="required"/>
      <attribute name="type" type="string"/>
    </extension>
  </simpleContent>
</complexType>
```

### 7.6.7 Other Content Models

Instead of sequence, other content models may be used:

- *choice* means that only one of the children may appear.
- *all* means that each child may appear or not, but at most once each.

Consult the specification for more detail on these and other content models.

### 7.6.8 Schema Namespaces

Every schema should define a namespace for its elements and for internal references to types. For example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.uct.ac.za"
  xmlns:uct="http://www.uct.ac.za">

  <element name="author" type="uct:authorType"/>

  <complexType name="authorType">
    <simpleContent>
      <extension base="string">
        <attribute name="email" type="string" use="required"/>
        <attribute name="office" type="number" use="required"/>
        <attribute name="type" type="string"/>
      </extension>
    </simpleContent>
  </complexType>

</schema>
```

### 7.6.9 Schema Example

Here is an example of a full Schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.uct.ac.za"
  xmlns:uct="http://www.uct.ac.za" elementFormDefault="qualified"
  attributeFormDefault="unqualified"
>
  <complexType name="authorType">
    <simpleContent>
      <extension base="string">
        <attribute name="email" type="string" use="required"/>
        <attribute name="office" type="integer" use="required"/>
        <attribute name="type" type="string"/>
      </extension>
    </simpleContent>
```

## XML

```
</complexType>

<complexType name="versionType">
  <sequence>
    <element name="number">
      <simpleType>
        <restriction base="string">
          <pattern value="[1-9]\.[0-9]+">/>
        </restriction>
      </simpleType>
    </element>
  </sequence>
</complexType>

<complexType name="uctType">
  <sequence>
    <element name="title" type="string"/>
    <element name="author" type="uct:authorType"/>
    <element name="version" type="uct:versionType"/>
  </sequence>
</complexType>

<element name="uct" type="uct:uctType"/>

</schema>
```

Here is a valid XML example for the above Schema

```
<uct xmlns="http://www.uct.ac.za"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.uct.ac.za
    http://www.husseinsspace/teaching/uct/2003/csc400dl/uct.xsd"
>

  <title>test XML document</title>
  <author email="pat@cs.uct.ac.za" office="410"
    type="lecturer">Pat Pukram</author>
  <version>
    <number>1.0</number>
  </version>
</uct>
```

## Activity 2: Schema

Write a Schema for the following XML document.

```
<article xmlns="http://article.com">
  <name>Fermat's Last Theorem</name>
  <date>20010112</date>
  <length unit="pages">11</length>
  <author>
    <first>Jonathan</first>
    <last>Smith</last>
  </author>
  <author>
    <first>Mary</first>
    <last>Carter</last>
  </author>
</article>
```

## 7.7 Data and Metadata

Data refers to digital objects that contain useful information for information seekers. Metadata refers to descriptions of these objects. Many systems manipulate metadata records, which contain pointers to the actual data.

### 7.7.1 Metadata Standards

To promote interoperability among systems, there are popular metadata standards to describe objects (both semantically and syntactically).

- **Dublin Core:** uses fifteen simple elements to describe every object.
- **MARC:** a comprehensive system devised to describe items in a (physical) library.
- **RFC1807:** the computer science publications format.
- **IMS Metadata Specification:** courseware object description.
- **VRA-Core:** multimedia (especially image) description.
- **EAD:** aids to locate archived items.

#### Dublin Core Example

Dublin Core is one of the most popular (and simplest) metadata formats. It contains fifteen elements, each with recommended semantics. All the elements are optional and repeatable. They are:

Title	Creator	Subject
Description	Publisher	Contributor
Date	Type	Format
Identifier	Source	Language
Relation	Coverage	Rights

Below is a Dublin Core in XML example:

```
<oaic:dc xmlns="http://purl.org/dc/elements/1.1/"
xmlns:oaic="http://www.open
  <title>02uct1</title>
  <creator>Hussein Suleman</creator>
  <subject>Visit to UCT </subject>
  <description>the view that greets you as you emerge from the tunnel
  under th
  <publisher>Hussein Suleman</publisher>
  <date>2002-11-27</date>
  <type>image</type>
  <format>image/jpeg</format>
  <identifier>http://www.husseinsspace.com/pictures/200230uct/02uct1.j
  pg
</identifier>
  <language>en-us</language>
  <relation>http://www.husseinsspace.com</relation>
  <rights>unrestricted</rights>
</oaic:dc>
```

### 7.7.2 Metadata Transformation

To do this, take the following steps:

1. Use an XML parser to parse data.

2. Use SAX/DOM to extract individual elements and generate the new format.

The following code converts UCT to Dublin Core (Don't worry if you do not understand it):

```
my $parser = new DOMParser;
my $document = $parser->parsefile ('uct.xml')->getDocumentElement;
foreach my $title ($document->getElementsByTagName ('title'))
{
    print "<title>".$title->getFirstChild->getData."</title>\n";
}
foreach my $author ($document->getElementsByTagName ('author'))
{
    print "<creator>".$author->getFirstChild->getData."</creator>\n";
}
print "<publisher>UCT</publisher>\n";
foreach my $version ($document->getElementsByTagName ('version'))
{
    foreach my $number ($version->getElementsByTagName ('number'))
    {
        print "<identifier>".
            $number->getFirstChild->getData."</identifier>\n";
    }
}
}
```

As you will see later in this unit, there is an easier way to achieve this in the unit.

## 7.8 XPath

The XML Path Language (XPath) supplies a mechanism to address particular nodes or sets of nodes in an XML document. XPath expressions can be used to write precise expressions to select nodes without using procedural DOM statements. For example, we can address particular nodes using expressions like:

```
uct/title uct/version/number uct/author/@office
```

### 7.8.1 XPath Syntax

- Expressions are separated by "/".
- In general, each subexpression matches one or more nodes in the DOM tree.
- Each sub-expression has the form: *axis::node[condition1][condition2]...* where *axis* can be used to select children, parents, descendants, siblings, and so on.
- Symbols may be used for the possible axes:

Expression	What it selects in current context
title	"title" children
*	All children
@office	"office" attribute
author[1]	First author node
/uct/title[last()]	Last title within uct node at top level of document
//author	All author nodes that are descendant from top level
.	Context node

## XML

..	Parent node
version[number]	Version nodes that have "number"
version[number='1.0']	Version nodes for which "number" has content of "1.0"

## 7.9 XSL

The XML Stylesheet Language (XSL) converts structured, XML files into a "human-friendly" representation. The thought underlying this is that, besides for programmers, no one should ever have to read or write XML. Conversions can be done in two steps:

1. Transform XML data
2. Process the data and stylesheet.

In systems that are Web-based, the first step is more useful — the first step being called XSL Transformations (XSLT) — as XHTML is directly "processed" by browsers.

## 7.10 XSLT

XSLT is a declarative language, written in XML, that specifies transformation rules for XML fragments. XSLT can convert any arbitrary XML document into XHTML or another XML format (e.g., different metadata formats). For example, the author tag can be converted as follows:

```
<template match="uct:author">
  <dc:creator>
    <value-of select="." />
  </dc:creator>
</template>
```

### 7.10.1 XSLT Templates

Templates of replacement XML are specified, with matching criteria, using XPath expressions. XSLT processors attempt to match the root XML tag with a template. If this fails they descend one level and try to match each of the root's children, and so on. In the previous example, all occurrences of the "uct:author" tag will be replaced by the contents of the template. Special tags in the XSL namespace are used to perform additional customisation, for example, *value-of*.

### 7.10.2 XSLT Special Tags

- **value-of, text, element:** Create nodes in result document.
- **apply-templates, call-template:** Explicitly apply template rules.
- **variable, param, with-param:** Local variables and parameter passing.
- **if, choose, for-each:** Procedural language constructs.

### 7.10.3 XSLT Language

*value-of* is replaced with the textual content of the nodes identified by the XPath expression. For example:

```
<value-of
  select="uct:title" />
```

## XML

*text* is replaced by the textual content. Plain text is usually sufficient. For example:

```
<text>1.0</text> 1.0
```

*element* is replaced by an XML element with the indicated tag. Usually the actual tag can be used. Example: `<element name="dc:publisher">UCT</element>` `<dc:publisher>UCT</dc:publisher>`

*apply-templates* explicitly applies templates to the specified nodes. Example: `apply-templates select="uct:version"/>`

*call-template* calls a template in a similar way to calling a function. This template may have parameters and must have a *name* attribute instead of a *match*. Example: `<call-template name="doheader"> <with-param name="lines">5</with-param> </call-template> <template name="doheader"> <param name="lines">2</param> ... </template>`

*variable* sets a local variable. In XPath expressions, a \$ prefix indicates a variable or parameter instead of a node. Example: `<variable name="institution">UCT</variable> <value-of select="$institution"/>`

Selection and iteration examples: `<if test="position()=last()">...</if>` `<choose>`  
`<when test="$val=1">...</when>` `<otherwise>...</otherwise>` `</choose>` `<for-each`  
`select="uct:number">...</for-each>`

### 7.10.4 XSLT Example

```
<stylesheet version='1.0' xmlns='http://www.w3.org/1999/XSL/Transform'
  xmlns:oaidc='http://www.openarchives.org/OAI/2.0/oai_dc/'
  xmlns:dc='http://purl.org/dc/elements/1.1/'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:uct='http://www.uct.ac.za'
>

<!--
  UCT to DC transformation Hussein Suleman
  v1.0 : 24 July 2003
-->

<output method="xml"/>

<variable name="institution"><text>UCT</text></variable>
<template match="uct:uct">
  <oaidc:dc
    xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
      /oai_dc/
      http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
    <dc:title><value-of select="uct:title"/></dc:title>
    <apply-templates select="uct:author"/>
    <element name="dc:publisher">
      <value-of select="$institution"/>
    </element>
    <apply-templates select="uct:version"/>
  </oaidc:dc>
</template>

<template match="uct:author">
  <dc:creator>
    <value-of select="."/>
  </dc:creator>
</template>

<template match="uct:version">
  <dc:identifier>
```

## XML

```
<value-of select="uct:number" />
</dc:identifier>
</template>
```

```
</stylesheet>
```

This is not the simplest XSLT that solves the problem. The transformed XML looks like this:

```
<?xml version="1.0"?>
<oaiddc:dc xmlns:oaiddc="http://www.openarchives.org/OAI/2.0/oai_dc/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:uct="http://www.uct.ac.za"
  xsi:schemaLocation=
    "http://www.openarchives.org/OAI/2.0/oai_dc/
    http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
  <dc:title>test XML document</dc:title>
    <dc:creator>Pat Pukram</dc:creator>
  <dc:publisher
    xmlns:dc="http://purl.org/dc/elements/1.1/">UCT</dc:publisher>
    <dc:identifier>1.0</dc:identifier>
</oaiddc:dc>
```

## 7.11 Answers

### 7.11.1 Answer to Activity 1

One possible DTD is:

```
<!DOCTYPE id_data [
  <!ELEMENT id_data (name, date_of_birth, blood_group?)>
  <!ELEMENT name (firstname, middlename*, lastname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT middlename (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ATTLIST date_of_birth day CDATA #REQUIRED>
  <!ATTLIST date_of_birth month CDATA #REQUIRED>
  <!ATTLIST date_of_birth year CDATA #REQUIRED>
  <!ELEMENT blood_group (#PCDATA)>
]>
```

# Chapter 8. Style Sheets

## Table of Contents

Objectives .....	1
8.1 Introduction to Style Sheets .....	1
8.1.1 Advantages of Style Sheets.....	1
8.1.2 Disadvantages of Style Sheets .....	2
8.2 CSS .....	2
8.3 Important Note About Rules .....	3
8.4 In-line Styles .....	3
8.5 Embedded Style sheets.....	4
8.6 Imported Style Sheet.....	4
8.7 Classes .....	5
8.8 Cascading Style Sheets .....	6
8.9 Review Question .....	8
8.9.1 Review Question 1: Reflection on Style Sheets.....	8
8.10 Discussions and Answers.....	8
8.10.1 Discussion of Exercise 1.....	8
8.10.2 Answer to Exercise 2 .....	9
8.10.3 Answer to Exercise 3 .....	9
8.10.4 Solutions to Exercise 4 .....	9
8.10.5 Solutions to Exercise 5 .....	10
8.10.6 Solutions to Activity 1 .....	11
8.10.7 Discussion of Review Question 1 .....	12

## Objectives

At the end of this chapter you will be able to:

- Explain the advantages and disadvantage of using stylesheets;
- Use CSS to create web pages.

## 8.1 Introduction to Style Sheets

There is no format to follow for teaching the aesthetics of style - most people, though, can recognise something that follows a classical design. But some things can be said about the style of a website. For instance, when Web pages belong to the same website, each page should have a consistent look in order to provide familiarity for the user.

Style sheets (sometimes referred to as templates) are used in desktop publishing to provide consistency when formatting text. The format applied by the stylesheet could be to indent every first line of a paragraph by 2cm, insert a page break at the end of every chapter, and so on. Naturally, due to multimedia, Web pages not only have to consider text formatting, but also visual and sound presentation, and various multimedia formats in general. Before we continue, let us briefly discuss the advantages and disadvantages of using style sheets.

### 8.1.1 Advantages of Style Sheets

1. **Multiple Styles** - A single document can be presented in multiple styles by using multiple style sheets.
2. **Re-styling** - The use of style sheets (which are separate to the HTML files) allows the quick re- styling of any document, without modifying the original HTML.

3. **Document maintenance** - The ability to re-style many documents allows us to easily make changes to the appearance of many Web pages without separately editing each one.
4. **Consistency** - Style sheets guarantee consistency throughout website.
5. **Optimal file size** - The smaller the files the faster the download. Using style sheets can help minimize file sizes, since, for example, every `<font>` tag, is defined in one place in a style sheet, rather than in multiple places in the HTML file.
6. **Style and structure** - When first developed, HTML was only concerned with document markup and not with the document's formatting. This eventually changed, with more and more functionality being added to HTML to allow for formatting. With the introduction of style sheets, the HTML document is again concerned only with structural document markup — all formatting is now placed in the style sheet.

### Exercise 1

Visit the UCT Computer Science Department website [<http://www.cs.uct.ac.za>] and suggest how the above advantages of style sheets can be used. You can find some discussion on this at the end of this chapter.

## 8.1.2 Disadvantages of Style Sheets

1. **Browser dependency** - Style sheets format things slightly differently on different browsers. Unfortunately, browsers have different support for HTML and style sheets. Newer browsers have largely converged on HTML support so that HTML documents look the same across different browsers. The state of style sheet support is somewhat worse, largely because style sheets are newer than HTML.
2. **Old Browsers** - Some very old browsers (such Netscape Navigator 2) do not support style sheets. All in all, style sheets have only minor disadvantages, and should be used when developing websites.

### To Do

Read up on style sheets on your text books. Can you see any advantages or disadvantages that have not been presented here? Also use the Internet to find out more about the usage of style sheets. A good starting place is the W3C's section on style sheets [<http://www.w3.org/Style/>].

## 8.2 CSS

The style sheet standard supported by modern browsers is called cascading style sheets, or CSS. CSS files contain a set of rules for the formatting of HTML documents. An example is given below:

```
<html>
<head>
<title>UCT MSc IT Example 1 on style sheets</title>
<style>
  BODY {

    font-family : "times new roman;
    margin-left : 20%;
    margin-right: 20%;
    text-align : justify;
    background : ivory;
    color : black;
  }

  P {
    text-indent : 2cm;
  }
</style>
</head>
```

A style sheet is a collection of rules that describe the format of the HTML tags. In the above example there are six

## Style Sheets

rules describing the format of the BODY tag, and one rule for the P tag. There are two ways:

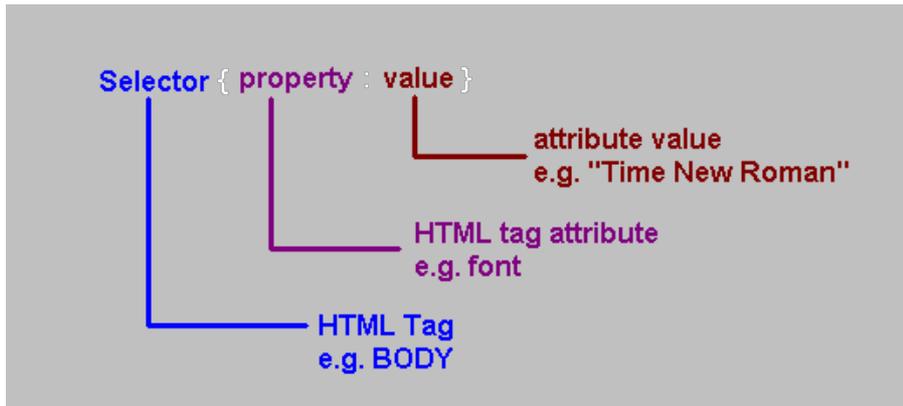
to write style sheets: the technically easier rule-based approach, and an approach that procedurally constructs a style sheet — such an approach is outside the scope of this unit, but feel welcome (if you any spare time) to visit various sites on this topics such as this one [<http://csgrs6k1.uwaterloo.ca/~dmg/dsssl/tutorial/tutorial.html>] or search on Google [<http://www.google.com>] for more

Below is a description of the rules used in the above example.

**Body** - *bgcolor* set to "ivory", left and right margins indented relatively by 20%, font set to "Times New Roman" and text colour set to black.

**P** to indent the first line by and absolute value of two centimeters.

There are three parts to a style sheet rule, shown in the figure below:



## 8.3 Important Note About Rules

Notice in the given example how each rule is separated by a semi-colon (;). If your code is not working, ensure that the semi-colons are present.

There are three ways to apply the above CSS rules to an HTML file:

1. add them in-line to the HTML file itself, attached directly to the relevant HTML tag.
2. embed the rules into the HTML file
3. link the CSS file to the HTML file

The next section will look at each of these methods.

### Exercise 2

Add a style sheet property to the given example to change the text color to dark red. You can find the solution at the end of the chapter.

## 8.4 In-line Styles

In-Line styles are added to individual tags and are usually avoided. Like the FONT tag they clog up HTML documents, making them larger and increasing their download times.

An example of an in-line style is given below:

```
<P style="text-indent: 2cm; color:darkred;">  
This paragraph has been formatted using the in-line style command.  
</P>
```

```
<P>
This paragraph has not been formatted using the in-line style
command.
>/P>
```

## 8.5 Embedded Style sheets

This method avoids duplication within a single HTML document. However, it still has its drawbacks: every Web page on your site needs this embedded style sheet inserted; consequently any updates to the style sheet have to be made to every HTML document that has the style sheet embedded in it. We have already used embedded style sheets as they are the simplest to implement, here is another example:

```
<html>
<head>
<title> University of Cape Town, Example on embedded style
sheets</title>
<sty
le>
BODY
{
font-family : "times new
roman; margin-left : 20%;
margin-right:
20%; text-align
: justify;
background :
ivory; color :
darkred;
}
P {
text-indent : 2cm;
}
h1,h2,h3{
color:red
; margin-
left:2cm
}
</style>
</head>

.
.
.

</html>
```

### Exercise 3

The above code has a deliberate mistake in it. Can you find it? You can find the corrected version at the end of the chapter.

Notice how rules for the h1, h2, and h3 tags can be simultaneously defined. Embedded style sheets should always be placed within the HEAD — preferably before the TITLE (i.e. not like in the above example).

## 8.6 Imported Style Sheet

This gives you all the advantages of style sheets: by changing a single value in one file the format

change is propagated to all of the HTML documents linked to the style sheet. The style sheet is written just as an embedded style sheet is, but, instead of inserting it in an HTML file, it is saved as a separate file (usually with a .css extension). Each HTML document then imports the CSS file. There are two ways to import a file:

### Linking it

```
< link rel="stylesheet" href="
  ../pathname//stylesheet_filename.css" type="tex
```

### Importing it

```
<style>
@import (http://pathname/stylesheet.css);
</style>
```

The second option will only work for HTML documents on a Web server. If this is not the case, use the first option.

## Exercise 4

From the in-line style listing below, create an embedded style sheet and a linked style sheet.

```
< h1 style="color:red;margin-left:2cm" >h1
heading</h1>
< h2 style="color:red;margin-left:4cm" >h2 sub-
heading</h2>
< p style="text-indent:2cm;color:darkred;margin-
left:6cm">Paragraph 1</p>
< p style="text-indent:2cm;color:darkred;margin-
left:6cm">Paragraph 2</p>
< h2 style="color:red;margin-left:4cm" >h2 sub-
heading</h2>
< p style="text-indent:2cm;color:darkred;margin-
left:6cm">Paragraph 1</p>
```

You can find a solution to this exercise at the end of this chapter.

## 8.7 Classes

Sometimes you may wish to give different formats to different paragraphs in the same HTML document. This can be accomplished by using classes.

A classic example is the formatting of publications in journals, where the leading paragraph is an abstract and has the following format:

**Title: 16pt, Arial, bold, centered**

**Authors: 14pt, Arial, bold, centered**

*abstract : 12 pt Times New Roman, italic, justified left, left margin 4cm, right margin 6cm first line indent 1cm*

**Section Headings: Arial 14pt, bold, first line indent 1 cm**

**Sub-Section Headings:Arial 12pt, bold, first line indent 1 cm**

body text: Times New Roman, 12 pt, first line indent 1 cm, left margin 3cm, right margin 3cm

There are also many different formats for various other paragraphs. Each different paragraph type can be given its own class name, such as "abstract". To do this, we use the P tag as shown in the code below. Open a new file and type in the following code and save it as csexercise.css

```
p.abstract{
margin-left:4cm;
margin-right:4cm;
font-style:italic;
font-family:"times New roman";
font-size : 12pt;
text-indent : 1cm;
text-align : left;
}
.body{
margin-left:3cm;
margin-right:3cm;
font-family:"times New roman";
font-size: 12pt;
text-indent: 1cm;
}
```

Open a blank document and import the above style sheet. The abstract class is used as shown in the code below:

```
<P CLASS="abstract">
```

### Exercise 5

As an exercise, add code to the stylesheet (csexercise.css) for the title, author, section heading and subsection heading. You can find the solution at the end of this chapter.

### To Do

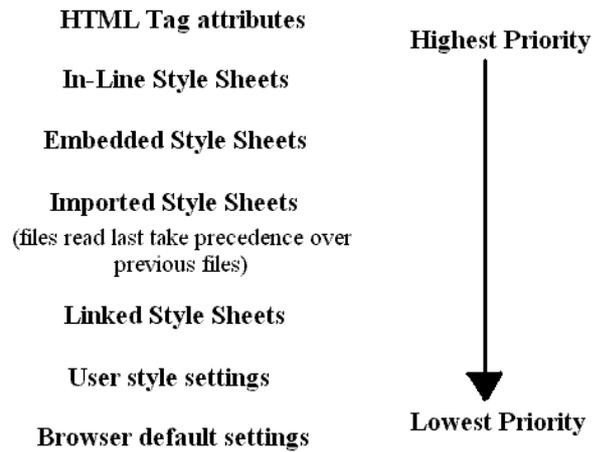
Visit Webmonkey's article "Inject Some Style (Sheets) into your HTML" [<http://webmonkey.wired.com/webmonkey/html/96/34/index2a.htm>]. Also visit the sites referred to at the end of the article.

## 8.8 Cascading Style Sheets

The cascading style sheet standard supplies very powerful tools to control Web page formatting. For instance, consider a university with many departments — each with their own individual design criteria — that is producing a website. It is possible to create a hierarchy of style sheets that allows each department's website to maintain formatting consistency with all the other university sites, while allowing each department to deviate from the format where needed.

The hierarchical (cascading) structure of style sheets can be used to do this. The figure below illustrates the style hierarchy design by W3C.

## Style Sheets



### To Do

Visit John Allsop's complete CSS Guide [[http://www.westciv.com/style\\_master/academy/css\\_tutorial/index.html](http://www.westciv.com/style_master/academy/css_tutorial/index.html)] for more information on style sheets.

### Activity 1

You are designing a Web page to display pop song lyrics (naturally the copyright has been permitted). The information needs to be displayed on the page in the style presented below. Use only the h1, h2, body and p HTML tags; note that two or more classes are needed for some of these tags.

- **Artist:** 14pt, Arial, Bold, Moccasin
- **Song Title:** 14pt, Arial, Normal, Khaki
- **Album:** 12pt, Arial, Normal, Khaki
- **Year:** 12pt, Arial, bold, ivory
- **Background:** colour=Chocolate
- **Verse:** 14pt, Trebuchet MS, Bold, gold, margin-left 2cm, margin-right 2cm
- **Chorus:** 14pt, Trebuchet MS, Bold & Italic, gold, margin-left 4cm, margin-right 4cm

A solution can be found at the end of the chapter.

### Activity 2

Use the style sheet from Activity 1 to create several Web pages that *import* the style sheet.

### Activity 3: Drop-cap lettering & Tables

A company wants you to design a similar website to that of Activity 1. This company wants the Web page to display their products prices in the format listed below:

Product description	product price
O ranges	7p
A pples	5p

The first letter example uses the span tag and is detailed below. Create a class fl with the following attributes:

```
span.fl{font-size:32pt;font-family:"brush script mt";}
```

Place the required letter between span tags using the above class, as shown below:

```
<span class="fl">A</span>pples
```

## 8.9 Review Question

### 8.9.1 Review Question 1: Reflection on Style Sheets

Read the following case study and give reasons for asking certain question in relation to style sheets.

A school wishes to design a website. The function of this website is to let its students know when assignments are due, the dates of exams, timetables, subjects taught, syllabuses and the teachers who teach them.

The school wishes to have quality controls over the Web design, since the teachers themselves will be responsible for updating the information on the Web.

1. The school has approached you to advise them on how this may be possible.
2. The school wants you to explain the process you wish to use, and why it is better than letting every teacher write their own style.
3. Write a list of ten or more HTML tags and classes that may be used for the website. Consider the attributes that may have to be used, such as background colour, departmental style, tables (for timetable, exam, curriculum and annual), headers and so on.
4. After choosing the HTML tags, consider the CSS code required to style them.

Look at the end of the chapter for a discussion of this review question.

## 8.10 Discussions and Answers

### 8.10.1 Discussion of Exercise 1

Style sheets can be used to make the formatting of the site more consistent. This consistency enables the

user to recognise that they are on the Computer Science site no matter what page they are on.

Style sheets also enables the site to be more easily maintained. For instance, if it was decided that the indentation was too large, the indentation on each of the site's pages can be changed by modifying just one document: the site's style sheet.

## 8.10.2 Answer to Exercise 2

```
P {
  text-
  indent :
  2cm; color
      :
  darkred;
}
```

## 8.10.3 Answer to Exercise 3

```
BODY {
  font-family : "times new
  roman"; margin-left :
  20%;
  margin-right:
  20%; text-
  align :
  justify;
  background :
  ivory; color
  : darkred;
}
P {
  text-indent : 2cm;
}
h1,h2,h3
{
  color:red;
  margin-
  left:2cm;
}
}
```

## 8.10.4 Solutions to Exercise 4

embedded:

```
<h1 style="color:red;margin-left:2cm">h1 heading</h1>
<h2 style="color:red;margin-left:4cm">h2 sub-heading</h2>
<p style="text-indent:2cm;color:darkred;margin-left:6cm">Paragraph
1</p>
<p style="text-indent:2cm;color:darkred;margin-left:6cm">Paragraph
2</p>
<h2 style="color:red;margin-left:4cm">h2 sub-heading</h2>
<p style="text-indent:2cm;color:darkred;margin-left:6cm">Paragraph 1 </p>
```

Linked - html file

```
<html>
<head>
<link rel="stylesheet" href="ex4.css" type="text/css">
</head>
<body>
<h1>h1 heading</h1>
<h2>h2 sub-heading</h2>
<p>div.abstract </p>
<p>Paragraph 1</p>
<p>Paragraph 2</p>
<h2>h2 sub-heading</h2>
<p>Paragraph 1</p>
</body>
</html>
```

Linked - ex4.css

```
p.abstract{
margin-left : 4cm;
margin-right : 4cm;
font-style : italic;
font-family : "times New roman";
font-size : 12pt;
text-indent : 1cm;
text-align : left;
}
.body{
margin-left : 3cm;
margin-right : 3cm;
font-family : "times New roman";
font-size : 12pt;
text-indent : 1cm;
text-align : left;
}
```

## 8.10.5 Solutions to Exercise 5

New csexercise.css

```
h1{
font-family : arial;
font-size : 16pt;
font-style : bold;
Text-align : center;
}

h2
{
font-family : arial;
font-size : 14pt;
font-style : bold;
Text-align : center;
}
```

```

h3.sectionHeading
{
font-family : times new roman;
font-size : 14pt;
font-style : bold;
Text-align : left;
text-indent : 1cm;
margin-left : 3cm;
margin-right : 3cm;
}

h3.sectionSubHeading
{
font-family : times new roman;
font-size : 12pt;
font-style : bold;
Text-align : left;
text-indent : 1cm;
margin-left : 3cm;
margin-right : 3cm;
}

div.abstract{
margin-left : 4cm;
margin-right : 4cm;
font-style : italic;
font-family : "times New roman";
font-size : 12pt;
text-indent : 1cm;
text-align : left;
}
.body{
margin-left : 3cm;
margin-right : 3cm;
font-family : "times New roman";
font-size : 12pt;
text-indent : 1cm;
text-align : left;
}

```

## 8.10.6 Solutions to Activity 1

```

body{background-color:chocolate;
font-family:"arial";
font-weight:bold;}

h1.artist{color:moccasin;font-size:14pt;}
h1.songtitle{color:khaki;font-size:14pt;font-
weight:normal;} h1.album{color:khaki;font-size:12pt;font-
weight:normal;} h1.year{color:ivory;font-size:12pt}

p.verse{font-family:"trebuchet ms";
color:gold;
margin-left:2cm;
margin-right:2cm;
font-weighth:bold;
}
p.chorus{font-family:"trebuchet ms";

```

```
font-weight:bold;
font-style:italic;
color:gold;
margin-left:2cm;
margin-right:2cm;
}
```

## 8.10.7 Discussion of Review Question 1

The following answers are a guide only — you may have listed answers not included here. If this is the case, start a discussion forum "Why Use style sheets?" and compare answers with other colleagues.

- You advise the school that cascading style sheets should be used. Your reasoning could include?
  - Cascading style sheets reflects the school's structure. In other words, the school and its various departments can each tailor the style sheet (using a hierarchy of style sheets) to suite their own image, thereby providing some control and consistency over the overall layout without being too restrictive on each department's Web page design.
  - If the school were to change its image in the future, style sheets provide a quick and easy way to do so.
  - If a design change is required during the development of the website, then style sheets provide a quick and easy method to do so.
- You mention that if every teacher were to write their own style, the school's image would be lost. Further, updating these separate designs to meet some future design standard is an arduous task.
- HTML tags:
  - p, for text presentations.
  - table.exam, for exam timetables.
  - table.time, for student timetables.
  - table.year, for yearly timetables.
  - body, the style for the main page, which will be inherited by other pages.
  - body.english, the style for the english department.
  - body.maths, the style for the mathematics department.
  - h1, headers.
  - h2, other headers.

# Chapter 9. Website Design

## Table of Contents

Objectives .....	2
9.1 Introduction .....	2
9.2 Website Design.....	3
9.2.1 Guidelines for Developing a Website.....	3
9.2.2 Analysing Overall Site Aims.....	3
9.2.3 Website Architecture.....	4
9.2.4 Navigation Planning.....	7
9.2.5 Designing for Hyper-Reading .....	8
9.2.6 Estimating Download Times.....	9
9.3 Formats of Web Graphic Images.....	10
9.3.1 Main Features of the GIF Format.....	10
9.3.2 GIF Headers .....	11
9.3.3 GIF Colour Palettes.....	11
9.3.4 Interlacing .....	12
9.3.5 Transparency.....	12
9.4 Features of JPEG Format.....	12
9.4.1 'Lossy' and 'lossless' compression techniques .....	13
9.5 Designing Website File Structure.....	15
9.6 Review Questions.....	16
9.6.1 Review Question 1 .....	16
9.6.2 Review Question 2 .....	17
9.6.3 Review Question 3 .....	17
9.6.4 Review Question 4 .....	17
9.6.5 Review Question 5 .....	17
9.6.6 Review Question 6 .....	17
9.6.7 Review Question 7 .....	17
9.6.8 Review Question 8 .....	17
9.6.9 Review Question 9 .....	17
9.6.10 Review Question 10.....	17
9.6.11 Review Question 11 .....	18
9.6.12 Review Question 12.....	18
9.7 Discussion and Answers.....	18
9.7.1 Discussion of Activity 1: Bad Websites.....	18
9.7.2 Discussion of Activity 5: Using GIF Files.....	18
9.7.3 Discussion of Activity 6: Image Sizes.....	20
9.7.4 Answer to Review Question 1.....	20
9.7.5 Answer to Review Question 2.....	20
9.7.6 Answer to Review Question 3.....	21
9.7.7 Answer to Review Question 4.....	21
9.7.8 Answer to Review Question 5.....	21
9.7.9 Answer to Review Question 6.....	21
9.7.10 Answer to Review Question 7.....	21
9.7.11 Answer to Review Question 8.....	21
9.7.12 Answer to Review Question 9.....	21
9.7.13 Answer to Review Question 10.....	21
9.7.14 Answer to Review Question 11.....	21
9.7.15 Answer to Review Question 12.....	21

# Objectives

At the end of this chapter you will:

- Understand some basic global trends in e-commerce;
- Understand basic guidelines for website design;
- Understand some of the common mistakes in website design;

## 9.1 Introduction

There are at least four design issues that must be addressed when developing a website for e-commerce:

1. choosing the user interface for the content;
2. choosing the required images;
3. choosing the appropriate technology; and
4. devising the directory and file structures that will allow for the development and maintenance of the site.

The first aspect belongs to the discipline of human-computer interaction (HCI) and the second to graphic design. This course does not cover graphic design, but will mention some of the computer graphics technologies involved. The chapter examines elements of HCI as it applies to the Web in order to explore how websites can be made more usable, especially for potential customers.

The fourth choice does not directly affect the user. It is concerned with the organisation of the HTML files, the graphics files (e.g. JPG files), etc, on the server. Of course, a failure to properly organise these components of a website can lead an inability for the site and its developers to quickly respond to changing business priorities and circumstances.

Internet commerce has become a major part of business. A report by the International Telecommunication Union indicates that globally, 3.2 billion people are using the Internet, of which two billion live in developing countries<sup>1</sup>. Further, the report indicates that between 2000 and 2015, Internet penetration has increased almost seven-fold from 6.5 to 43 per cent of the global population. The proportion of households with Internet access at home advanced from 18 per cent in 2005 to 46 per cent in 2015<sup>1</sup>. Much of the growth in web connectivity has come from mobile. Mobile broadband penetration has gone up 12-fold since 2007<sup>2</sup>. A 2015 study by Price Waterhouse Coopers (PWC) indicated that many countries are seeing more online shopping than ever before. It also revealed that while behaviors vary by country, consumers around the globe share the same expectations when it comes to online shopping — they want efficiency, convenience and enjoyment<sup>3</sup>. The report also indicated global online shopper behavior by country. For example, Figure 4.1 indicates the online shopping behavior of South Africans. The survey indicates that South African residents are the most willing to pay for same-day deliver (79%)<sup>3</sup>. On a global scale, the products that are purchased the most online are books, movies, music and video games. The study revealed that 63 percent of shoppers said they prefer to purchase these items online, while 31 percent said they would rather buy from a store<sup>4</sup>. Recognizing these trends, browsers running on personal computers are no longer assumed to be the only means for potential customers to access a site. They are connecting via dedicated Internet appliances, such as phones with built-in browsers, via interactive TV, via hand held devices, etc.

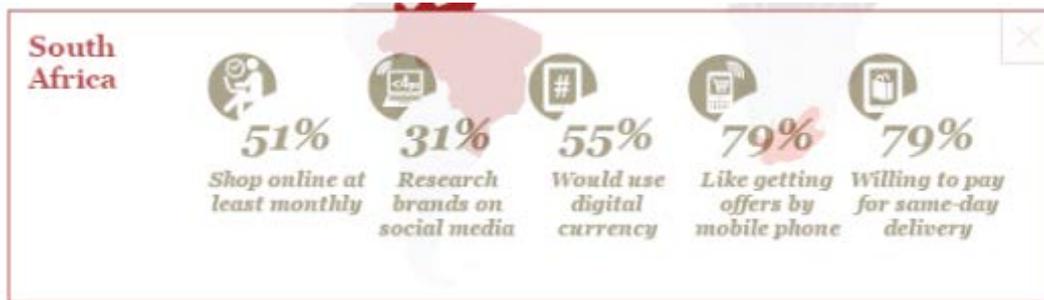
---

<sup>1</sup> [http://www.itu.int/net/pressoffice/press\\_releases/2015/17.aspx#.VmEaVfkrLIV](http://www.itu.int/net/pressoffice/press_releases/2015/17.aspx#.VmEaVfkrLIV)

<sup>2</sup> <http://time.com/money/3896219/internet-users-worldwide/>

<sup>3</sup> <http://www.pwc.com/gx/en/industries/retail-consumer/global-multi-channel-consumer-survey/survey-highlights.html>

<sup>4</sup> <http://www.pwc.com/gx/en/industries/retail-consumer/global-multi-channel-consumer-survey/retail-subsectors/books.html>



**Figure 4.1: Online shopping behavior of South African residents (Source: PWC)**

A website designer must, therefore, guarantee that consumers will:

- find the business website that they are developing;
- consider it interesting and attractive;
- find it easy to use; and
- be encouraged to return.

How do we, as developers of Internet commerce websites, attain these goals? To answer this, we begin by examining an approach to website design, and follow that by highlighting what a designer should not be doing.

## 9.2 Website Design

Unlike with software development, there are no accepted, well organised and documented methods for developing Web sites. There are, however, some good practices — for example, it is best to avoid 'technology traps' by not committing to new, but untried, technologies.

### 9.2.1 Guidelines for Developing a Website

The development of many websites is driven by the enthusiasm of designers and implementers — people who are often keen to use the latest technology. Being driven by technology rather than business and customer needs can lead to sites that are slow in execution, complicated to use, and do not achieve the intended results. Ironically, committing to a new technology too early can cause a website to rapidly look dated, either because the technology did not become popular and so stayed in an old form, or because the adoption of the new technology was so expensive that little budget remained for site maintenance.

As a general rule, avoid the unnecessary use of technology. While doing so, also recognise that any technology that has been avoided at some point in a website's lifetime may become vital to the site as the site's needs change, or even as user interface fashion progresses. This idea of change, and the fluidity of the guidelines, is important. Technology that is unnecessary today may be acceptable or even necessary in a few years' time. For example, it was once considered inadvisable to use frames or scrolling text. Now, these are less of an issue: frames are supported by most browsers in current use, and users are used to scrolling. (However, we will still discuss these below.)

### 9.2.2 Analysing Overall Site Aims

The first activity of website development is the definition of the site's purpose. Begin with the broadest idea and refine this.

Once the broad, high-level description of the site's purpose is known, refine the goals in more detail. Ask yourself, "What are we trying to achieve?" and answer the question in a way that adds detail to the description. Do not only consider the obvious questions concerning the site's purpose, but also the issues around the implicit messages a potential customer might receive from the site. For example, if a website selling holiday packages offers most of its packages in, say, Italy, as that is where most of the company's customers have traditionally wanted to go, then the site might give the impression that Italy is the only country that the company knows about. While some messages should be avoided, there might also be positive messages that the site should give, such as those concerning the quality of the service, or how long the company has been in business.

## Website Design

Thus, one aspect of website design is to identify primary and secondary goals for the site. The primary goals may be achieved by directly implementing facilities to meet those goals, for example, by providing a catalogue of goods and services. The secondary goals may be achieved in more subtle ways, by, for example, choosing colours that suggest stability, excitement, or so on; or by providing subtle, positive messages to the customer concerning the company running the website and the services on offer.

Let us pursue these ideas with an example. Say you are in the business of developing software on behalf of clients, and that you also sell programming language compilers and development environments. Your clients might choose both the programming language and a specific compiler for it. For instance, one client might choose Java, and another might choose C++. The second of these clients may or may not have an opinion as to whether Microsoft Visual C++ should be used in preference to IBM's VisualAge C++. You sell both, as well as other C++ development systems, so you can cope with the choice of either. You have decided that the main purpose of your new Web site is to 'sell the services' of your company. How do you determine the details of this? One method is brainstorming, in which a group of people briefly consider all aspects of an idea. Participants in a brainstorming session are expected to contribute to discussions, to arguments in an unstructured, anarchic, often unruly way. They will often jump from one idea to another, calling out words, arguing for and against points — sometimes simultaneously. In brainstorming the ideas of all the participants are initially given equal weighting, so the most lowly employee may argue freely with the most exulted. Brainstorming is usually thought of as a fun way to generate ideas; it is a very creative process and the outcomes can often be surprising with unexpected benefits.

As can be seen from the previous exercise, expressing many ideas using brainstorming or a similar technique, may result in a mixture of ideas concerning the purpose of the site and how to achieve it. Ignore the detail of the 'how' and try to understand what it means. One suggestion of, say, using grey to suggest reliability may contradict a suggestion of using bright colours to convey excitement and being a 'go-getting' company, but that does not matter at this stage. The important information here is that both aspects of the company are to be conveyed. The question is whether it should be a primary or a secondary goal.

One question concerning primary goals that the brainstorming did not settle is that of what the site is primarily selling — software development skills or programming language implementations. The sooner this is answered the better — but it might require debate, and the decision may, to some extent, depend on what can be achieved in the time and budget available for site development. For example, it may be difficult to organise the selling of programming language implementations because the website might have to interact with a database of information about stock levels. Another reason for not promoting that side of the business might be that the company could not cope with too much extra business of that kind. The irrelevance of distance when selling products might increase sales of compilers and programming environments, whereas the need for software developers to meet their clients might naturally limit take-up if that side of the company's business was promoted.

As well as the website developers knowing what the site's purpose is, they must also know its audience. With knowledge of your audience, you can tailor both content and presentation to suit their needs and keep Web customers returning for more business.

Ultimately, knowledge of the business requirements and expected audience is the starting point of understanding the site requirements and design. Knowing these facets provides a foundation for site development. Such a foundation must be the basis for the design. It should drive the design. You must not start with an arbitrary design and force other choices to fit it.

### 9.2.3 Website Architecture

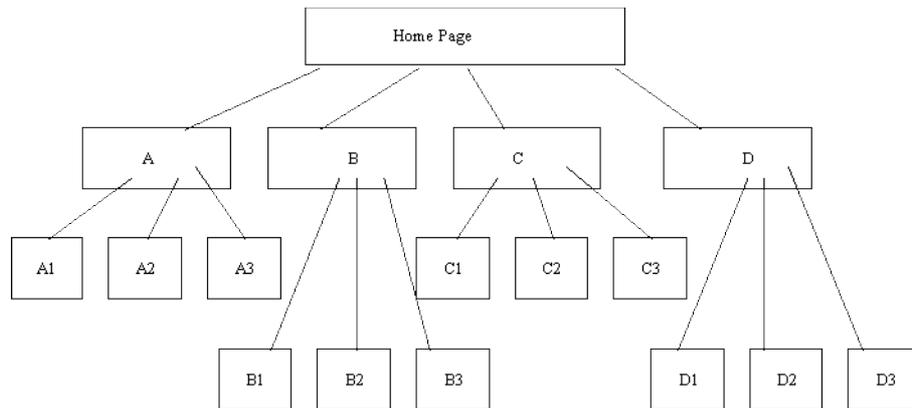
The design of a site involves many considerations, not least of which is navigation, a discussion of which occurs later. Other than navigation, the following points need to be considered as well:

- the structure of the site — how the pages are organised (logically, rather than physically on disk);
- the possible paths through the structure — both the paths you might prefer or expect a user to follow, and the paths you should provide to make the structure truly hypertext;
- the style — the so-called 'look and feel', that is, the layout of the actual pages;
- the structure of individual pages — how each page is organised (or how categories of page are

## Website Design

organised).

All the above points are related to each, and so it is not possible to prescribe an order in which they should be considered. For example, the structure of the site is related to the possible paths through the site, as will the structure of the individual pages. All should be strongly influenced by the site's purpose and target audience. Now let us consider an abstract view of how a site is organized. Figure 4.2 shows this structure. A structure such as Figure 4.2 is sometimes called an information architecture. The structure implies that users start at the home page, then visit pages A1, A2, and A3; followed by visiting pages B1, etc., through D3. This sequence of page visits, however, may not be the sequence the user chooses to view. Say the page organisation reflected the departments of an office equipment supplier — desks, filing cabinets, pens, paper, computers, fax machines, and the like.



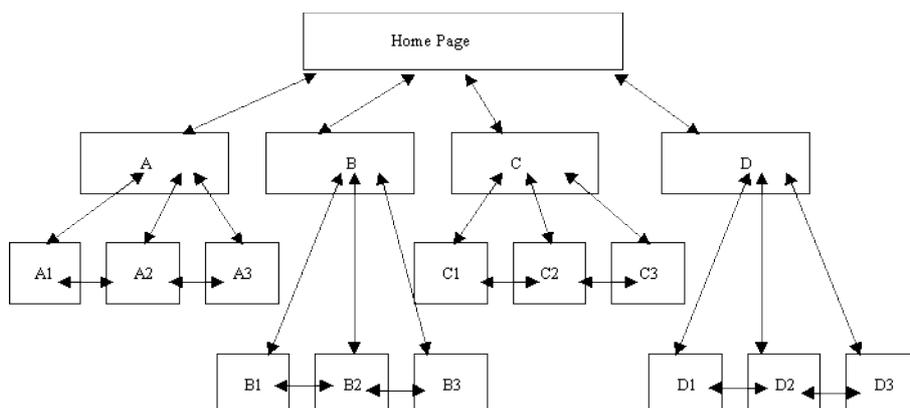
**Figure 4.2 Website information architecture**

Let us represent these departments as A, B, C, D as in:

- A represents the furniture department;
- B is for electronic (non-computer) equipment, such as fax machines;
- C is computers and associated items such as, scanners, printers, disk drives, etc; and
- D is copying services, including the production of brochures, business cards, etc.

Clearly, the information architecture is not devised with navigation primarily in mind. Let us briefly consider navigation: imagine a scenario in which a user wishes to buy a computer, a combination fax / scanner / printer, and furniture to hold these. From the home page the user might visit C, for computing equipment, then perhaps move between pages C1 and C2 as they decided which computer and fax / scanner / printer they want. Having bought the computer equipment, the user might want to go directly to A1 for a desk, then A3 for a chair, and finally A2 for a monitor stand. The diagram above does not represent this scenario or even the possible navigation paths. It only present a 'logical' view of how the pages in the website are related to each other.

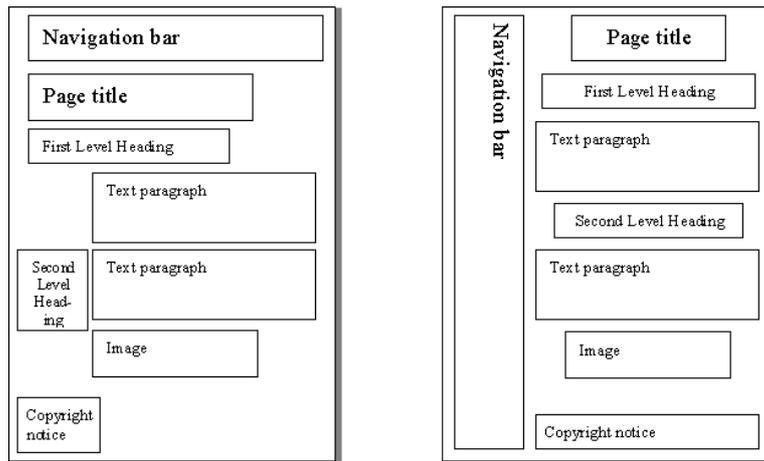
Now examine a second version of the site structure, with navigation paths added:



**Figure 4.3 Website information architecture with navigation paths**

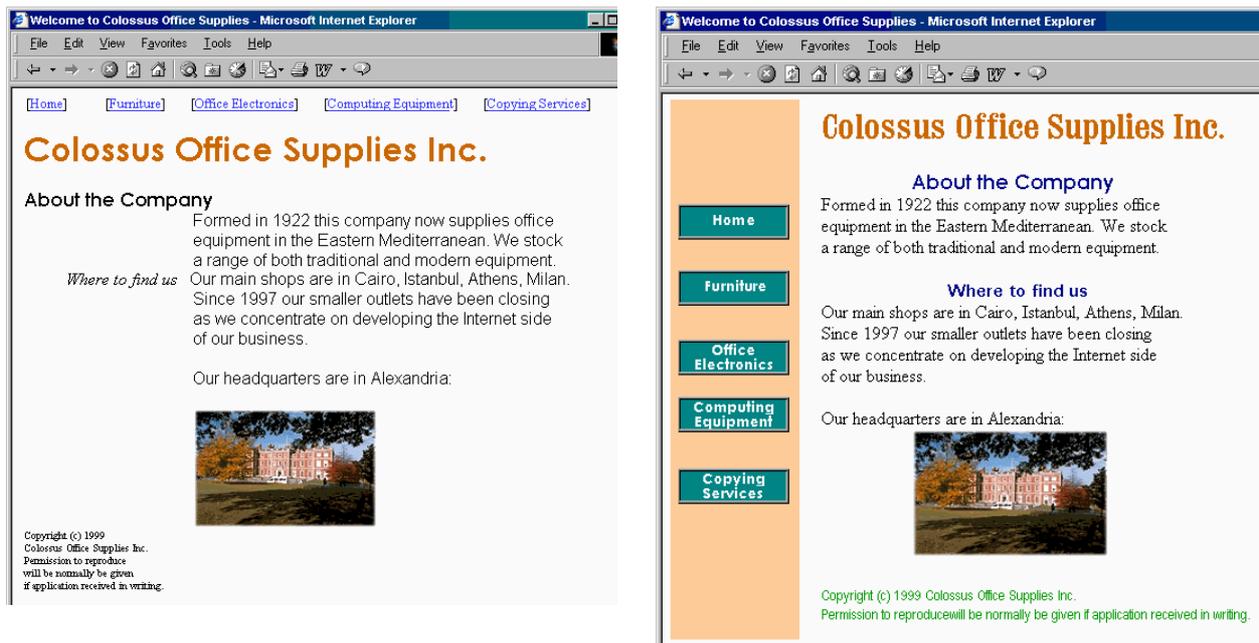
In this second version of the information architecture, explicit navigation paths have been given, each of which is a two-way path. This is based on the assumption that a user of the site would need to move backwards and forwards between 'sub-departments'. Notice that no navigation has been included between departments, between A and B, C, or D. This is an arbitrary decision, which in reality would be made knowing the site's purpose and audience. Navigation planning will be discussed further later in this chapter.

Having come up with a plausible information structure, the structure of the individual pages need to be designed — this is called the Web page architecture, and another design effort is required for this. This should be done before settling on the layout and style. For example, Figure 4.4 shows two possible page architectures for displaying the same information, namely a page of book or report style.



**Figure 4.4 Page architecture for displaying a book or report style**

These structures can be implemented in different ways, inasmuch as the fonts, colours, button styles, etc. could be varied for each. These are important facets of style, usually decided by graphic designers. Software developers should be primarily interested in the structure of the page elements. When these have been decided, the graphic designer's 'look and feel' can be achieved using HTML, JavaScript, and so on. Figure 4.5 shows instances of the above page architectures. Both have the same content, but different layouts and styles. The HTML facilities required by each is quite different. For example, it may be convenient to use tables to implement the first layout and frames for the second (although neither have been implemented with either tables or frames). Design is all about making choices, as can be seen here.



**Figure 4.5** Websites showing different layouts

### 9.2.4 Navigation Planning

As observed in Chapter 1, the provision of navigation aids in hypertext documents is very important. Navigation aids prevent users from losing their place. When reading a book, people start at a particular page and read the subsequent pages in order. Perusing a catalogue might not be as linear an activity, but the page numbers, table of contents and the index provide the means to navigate backwards and forwards, and to jump to arbitrary pages. Hypertext documents do not follow this fashion for two reasons:

- the reader can arrive at any page, via a hyperlink; and
- a reader can read or interact with the pages in any order.

One technique for designing the navigation paths is to storyboard the tasks that the users might undertake when visiting the site. Storyboarding is a technique taken from cinematography: the story is developed by roughly sketching each scene — as well as the actions in each scene — on a board, and placing each image in the desired sequence. For Web site development each page does not need to be sketch out, but each scenario does need to be documented and analysed.

Ideally, a storyboard should be developed for all the pages in a complex website so that style can be compared with function. As with all these design activities, the storyboard may have to change — but it is easier to change a storyboard than the final product, and a storyboard acts as a good starting point to website design.

Navigational aids are an important ingredient for successful website design. Browsers themselves have a variety of navigation aids including Bookmarks (Favourites) and History lists, while the JavaScript language has facilities to access these. A website designer can make use of these facilities, but the page architecture should provide the navigation aids that are considered necessary.

Overcoming the 'Lost In Hyperspace' problem is a difficult problem to solve. It occurs when a user follows a series of links and discovers that they do not know where they are in a site or how to retrace their steps to a previous page. There has been much research into this phenomenon, but this appears to have been ignored by Web designers.

The 'Lost in Hyperspace' problem can be prevented, or lessened, by applying three rules: in general every page in

your website should include: where you are; where you have been; and where you can go For example, the Middlesex University academic home page, shown below implemented two of these rules. It shows where the user is by the down pointing double-headed arrow (↕) beside the word Academic. The other words on that navigation bar (preceded by the ► symbol) show what the designers see as major

## Website Design

navigational steps to other topics. By contrast, under the 'Academic' heading there are nine links, from Studying at Middlesex University through to 'What's New'. The user has left the mouse pointer over International Students, which has been highlighted by a rectangle outline. (You will study how to achieve such an effect a later chapter).

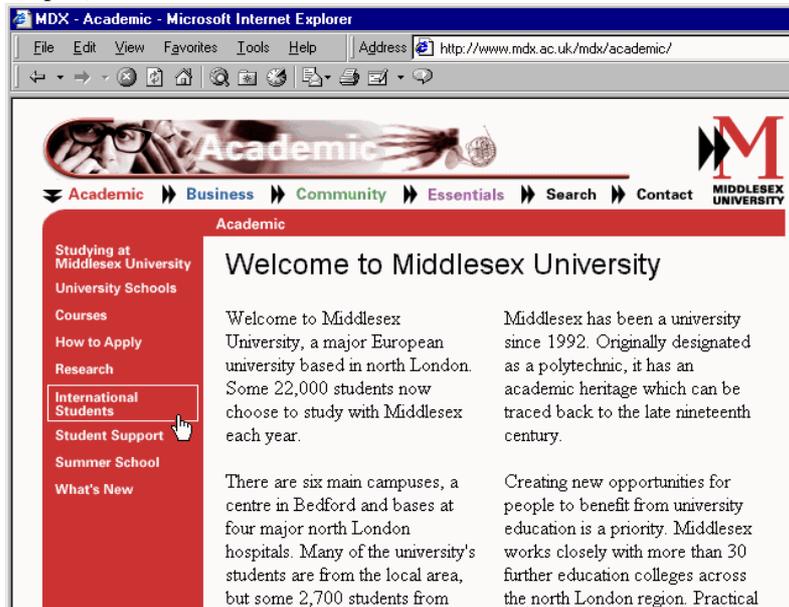


Figure 4.6 Middlesex University academic homepage

### 9.2.5 Designing for Hyper-Reading

As mentioned above, books are presented in a linear fashion while hypertext documents can be read non-linearly. The ordering of hypertext documents within a site is a related issue: visitors to a Web page may start in the 'middle', read through to the 'end' and finish at the 'beginning'. Within a single Web page, the top to bottom (and left to right) nature of the text may dominate, but if a user arrives at page somewhere other than at the document's top, there may be no order apparent to the reader. A Web designer must be sensitive to these many possible readings of a Web page, just as they should be sensitive to the navigational possibilities in an information architecture. The implicit ordering of a page architecture must be considered. Dynamically changing content (such as animations or JavaScript-produced content) will influence the order a user reads a page. This is something that does not happen in books.

Clearly, people adopt different reading manners between reading Web pages and books. In their 1997 study, Jakob Nielsen and John Morkes discovered that 79% of people merely scanned the text of a Web page. They found that only 16% read every word. Other work has found that less than 20% of readers scroll beyond the information that is visible. From these findings it could be concluded that Web pages should:

- contain 'scannable' text, in which keywords are highlighted to aid quick perusal;
- employ meaningful sub-headings;
- use bulleted lists to provide a quick summary of the main ideas;
- use only one idea per paragraph, concentrating on the introductory words of the paragraph;
- use the 'inverted pyramid' style of writing, where the document begins with the conclusion;
- use half or less of the word count of conventional writing; and
- ensure that all the information is visible on the screen without scrolling.

Such guidance should be used where needed, but should not be relied on slavishly. It is important to remember the intention behind the above suggestions. For example, from the third point, do not remember it as a statement to only 'use bulleted lists', but remember that the bulleted lists are meant to summarise information explained fully on the page. A study by Nielsen and Morkes emphasised the need for Web designers to instill in users a sense of the site's credibility, as well as a trust in the content provider: if readers do not believe what they read in any section of a page, they may skip the rest of the page. Credibility and trust are important to businesses wanting to establish commerce on the Net.

## Activity 1: Bad Websites

Visit the following websites:

- [www.websitesthatsuck.com](http://www.websitesthatsuck.com) [<http://www.websitesthatsuck.com>]
- [www.worstoftheWeb.com](http://www.worstoftheWeb.com) [<http://www.worstoftheweb.com> ]

Write down your view of these sites. Choose three of their featured websites and, for each chosen site, identify one aspect of its design that you find particularly poor. Try to find four or five examples of bad Web design and discuss them with colleagues on-line or in tutorial.

See Discussion on Activity 1 at the end of this chapter

## Activity 2: Top Ten Mistakes

Go to Jakob Nielsen's [useit.com](http://www.useit.com) website and read his original 1996 advice [<http://www.nngroup.com/articles/original-top-ten-mistakes-in-web-design/>] on what to avoid when designing a Web site. Then follow the link to his revised (1999) advice [<http://www.nngroup.com/articles/the-top-ten-web-design-mistakes-of-1999/>] and read it. Also have a look at some of the newer top-10 lists that can be found at the bottom of that page.

Take note of the main mistakes he lists and the extent to which he has changed his view. What about the other lists? Are there any inconsistency? Write down what you think of the current advice found on these lists.

## Activity 3: Design Issues

Read Nielsen's The Ten Most Violated Homepage Design Guidelines [<http://www.nngroup.com/articles/most-violated-homepage-guidelines/>]. Can you find an example of common websites that violates these guidelines? Also discuss whether or not you agree with the ranking.

Take note of the main mistakes he lists and the extent to which he has changed his view. What about the other lists? Are there any inconsistency? Write down what you think of the current advice found on these lists.

## Review

Do Review Questions 1, 2, 3, 4 and 5

## 9.2.6 Estimating Download Times

Hypertext research has shown that a user needs to be able to move between pages in less than a second for the site to have maximum effectiveness. Even with the increased use of fast modems, most pages remain painfully slow to download.

Factors in page download speed that are to a large extent beyond your control include:

- the throughput of the server;
- the server's connection to the Internet;
- the performance of the Internet itself;
- the user's connection to the Internet; and
- the rendering speed of the user's browser, computer processor and graphics hardware.

As far as possible, decisions that affect Web page speed should take this information into account. For example, if the website is to be used within a company as an intranet, or with limited access for registered customers as an extranet (see Unit 13), then the configuration of the users' computer systems should be completely known.

The main factor under a Web designer's control is the use of high-bandwidth information:

- Video and audio resources require the largest files; streaming technologies are offered by companies such as RealNetworks (via their 'real' media formats) and Apple (via QuickTime 4).
- Animation formats such as ShockWave (which include video and audio) require external applications or plugins in order for a browser to view them. This requires the designer to consider the large file sizes of the content, and interaction with external software.
- Graphics files may be large: JPEG is often used for high definition images and GIF for simpler ones. Various strategies may be employed to reduce the number of colours contained in an image (and hence reduce the file size, download time and the rendering time). For example, the distracting animation used earlier can take between 2 and 8 seconds to download depending on modem speed; this time can be reduced by half when colour reduction and other optimisations are used. The technologies involved are discussed later in this unit.

Apart from reducing image file sizes, a number of other techniques can be used to improve the apparent performance of a browser. For instance, browsers cache all the content that they download on the assumption that this content may be used again. So, a graphic used on a variety of pages can be reused from the cache rather than downloaded, as long as only a single copy of the image is used on the website. Using multiple copies of the image will circumvent any advantage that could be obtained from the browser's cache.

Specifying the size of the image to be displayed allows the browser to reserve the space needed for the image while the image is downloading; in the mean time, the rest of the page can still be displayed:

```
<img SRC="http://www.myCompany.com.au/general-images/ headquarters-  
photo.jpg" height=200 width=320>
```

There are also various 'tricks' that can be used to make download speeds appear faster. For example, an image can be downloaded twice — once on a page where the image will not be displayed (by setting the image width and height to zero), and then once again on a page where the image will be displayed. The browser will, when downloading the image for the first time, store it in its cache. Later, when the image should be shown, the browser will find the image in its cache, and so the image will appear to have downloaded much quicker than it truly has.

Netscape Navigator also provides a LOWSRC attribute for the <IMG tag; this allows for the designer to specify a file to load in advance of the true image. This should be a temporary, low-resolution, low colour, small image file — essentially a place holder for while the true image is being loaded.

## 9.3 Formats of Web Graphic Images

### 9.3.1 Main Features of the GIF Format

The GIF format was originally developed by CompuServe, who provided information of the kind now found on the Web. They provided email services before the Internet became popular and before the Web existed. They required a file format that could be used on different platforms (and which did not rely on specific hardware features) and supported some form of compression.

GIF files support any resolution (up to 65536 x 65536pixels). They provide indexed colour images supporting 8-bit colour information (256 colours). LZW (Lempel-Zev-Welch) compression is used to reduce file sizes. It works by compressing 'runs' of identical colour information. For example 100 contiguous red pixels in the same row take up 100 8bits, with LZW compression it is converted to 100 red pixels that takes up 16bits. LZW is a lossless compression technique, and so the compressed image is identical to the original image.

Although GIF met its original requirements, it does have its limitations: GIF allows a colour table of a maximum of only 256 colours, and LZW compression is a general technique not particularly efficient at compressing graphical images.

## 9.3.2 GIF Headers

GIF headers contain the information required by the client programme to display the image, as follows:

- the first 6 bytes denote the GIF version;
- GIF87a (the original 1987 definition);
- GIF89a (adds text overlays, multiple image overlay and transparent colour);
- 2 bytes for screen width;
- 2 bytes for screen height;
- 1 byte for colour information (0—2 palette size, 3 palette sorted, 4—6 colour depth, 7 global palette);
- 1 byte background colour;
- 3n global palette.

## 9.3.3 GIF Colour Palettes

### Adaptive Palettes

As an example of adaptive palettes, examine the image below and its full 255-colour palette. Each colour in the palette has an index which is used when specifying the colour. Index 40 has been selected (with the black square around it) in the palette below. Its colour is a brown made up of a Red value of 153, Green 102 and Blue 51.

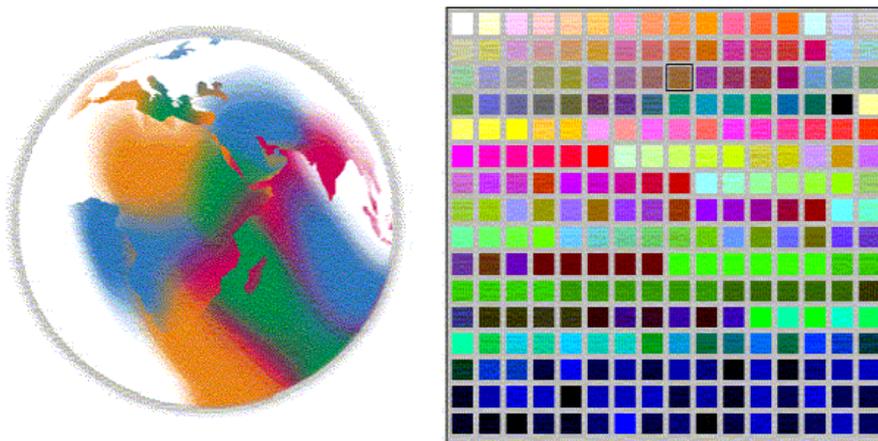


Figure 4.7 Adaptive palette with 255-colour palette

By contrast the almost indistinguishable image below has only 64 colours and a correspondingly smaller palette. A light green has been selected, with index 58 — Red 204, Green 255, Blue 204.

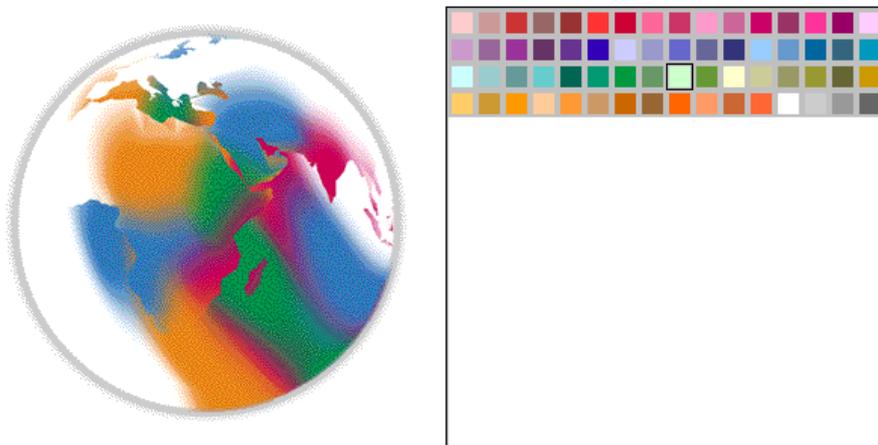


Figure 4.7 Adaptive palette with 64-colour palette

While adaptive palettes provide greater colour detail in images, they have some problems. Each image has a palette, which takes up extra space in a file. Furthermore, some machines can only deal with only one palette at a time, which results in palette clash if multiple palettes are used

### Fixed Palettes

Rather than creating a palette for each individual image, it is possible to use a standard palette, with the following pros and cons:

- The advantage is that the images will be smaller as no palette need be saved with each image;
- Conversely, this is quite a restriction on the image and is only suitable for 'drawn' rather than photographic images.

### 9.3.4 Interlacing

- Rather than storing the image in row order, GIF files can be stored with every 8th line at the head of the file, followed by every 4th, and so on.
- This file layout allows a low resolution version of the image to be displayed while the file is still downloading. As more of the file is downloaded, more detail can be displayed.

### 9.3.5 Transparency

With the 1989 GIF format, one colour in the colour table can be defined as transparent. This allows the background image/colour to show through, and for graphics to appear non-rectangular. However, allowing only one colour to be transparent can cause bad feathering effects to appear around the image's edge. For example, the world image was originally designed to be on a white background. This becomes clear when a Web page is given a non-white background colour, as in Figure 4.8 below. By designating white to be transparent, the background will show through, as in Figure 4.9.

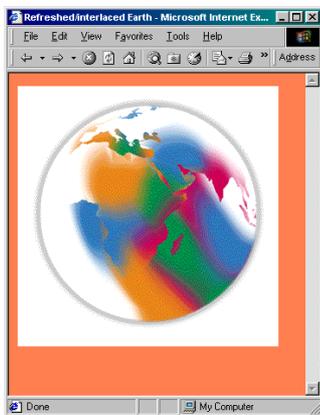


Figure 4.8 Non-white background

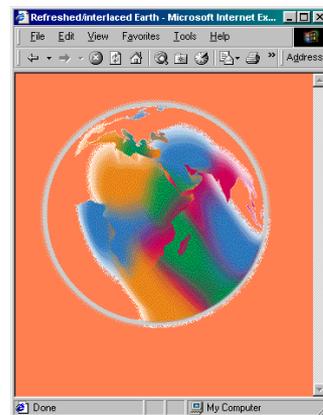


Figure 4.9 Transparent background

## 9.4 Features of JPEG Format

JPEG was designed to store photographic images. It employs a 'lossy' compression technique, in that it throws away information unlikely to be noticed by a human viewer. It allows the degree of compression, and hence the degree of degradation, to be specified (on a hundred point scale). The figure below is the dialogue box used by Adobe PhotoShop when saving a JPEG file.

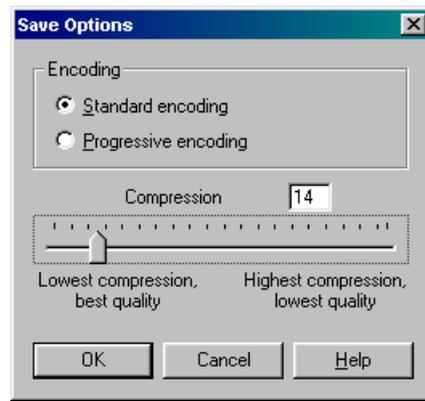


Figure 4.10 Photoshop dialog box

In photographic images with lots of variation in detail, degradation of this form does not matter — but it is noticeable for many types of images, especially those with sharply defined boundaries such as text. The figures below show enlarged versions of JPEG images saved with lowest compression and best quality (Q1); medium compression and quality (Q40); and maximum compression and lowest quality.

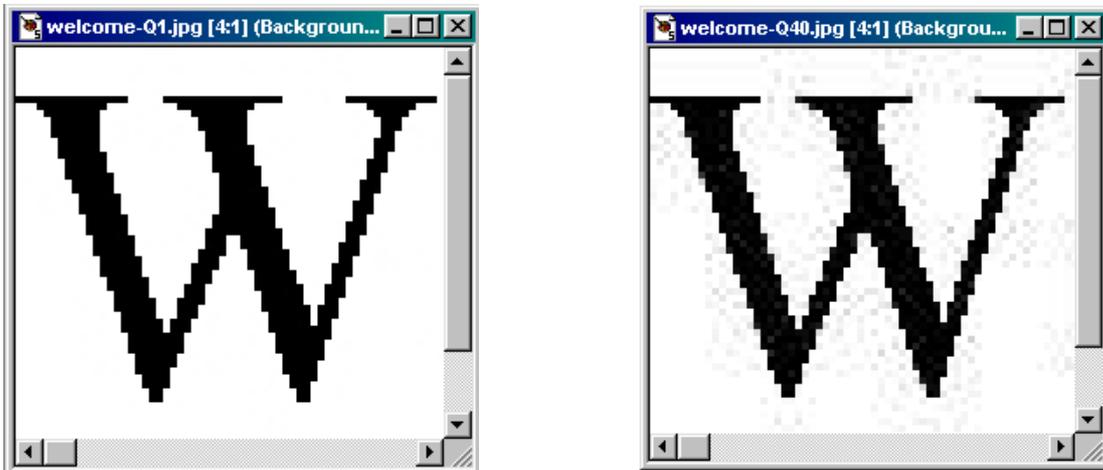


Figure 4.11 Enlarged versions of JPEG images with lowest and medium compressions



Figure 4.11 Enlarged versions of JPEG images with maximum compression

While lossy compression works well for photographs, line art (particularly text) blurs or 'ripples'. Because JPEG discards information, repeatedly editing and saving JPEG files causes a progressive worsening in the image's quality — this means that JPEG is usually only used for the final copy of the image, and not for any of the intermediate stages.

### 9.4.1 'Lossy' and 'lossless' compression techniques

To round off our discussion of the GIF and JPEG graphic formats, we now briefly consider the different approaches to compression and colour transformation.

### LZW compression

LZW compression, as used in the GIF format, builds a dictionary of common data sequences found in the image. It then replaces these sequences in the image with a short dictionary code. The dictionary need not be transmitted with the data, as it can be easily reconstructed. The algorithm is simple, and the technique is used in various file formats (not only graphic file formats), such as TIFF, DoubleSpace, Stacker, GZIP, ZIP and ARC. LZW is patented, but payments to the owners need only be made by those implementing the algorithm, not those using it. LZW compression works best on images with large, 'flat' areas of colour, as in cartoons, and drawings, rather than photographs.

### JPEG Colour Compression

JPEG colour compression exploits the human visual system. The compression technique begins by reducing the colour table. Most systems use an RGB (Red, Green, Blue) colour space, but this is difficult to compress. HSL (Hue, Saturation, Lightness) is better, as human eyes can more easily detect fluctuations in brightness than they can fluctuations in colour.

Using a technique called subsampling, the colour information in the colour table can be reduced while retaining lighting information (which is how black and white TV signals are sent).

### JPEG Colour Transformation

When looking at a far off image, all one can easily discern is the overall colour. Moving closer provides increasingly greater detail to be visible. JPEG uses a mathematical technique to replicate this effect. Treating each colour component (HSL) separately, each component is subjected to Discrete Cosine Transformations (DCT). This calculates the average colour of an area, as well as the rate at which the colour changes. These changes in colour are then quantized (rounded) so that sudden colour changes are removed but more gradual changes are retained (which results in the rippling artifacts, as the technique involves a cosine calculation).

### JPEG Compression

After applying the transformation to the colour table, the data is then compressed using traditional techniques. The most common of these is Huffman encoding, which works as follows:

- Calculate probabilistic occurrences of each character;
- Take the two characters with the lowest probabilities;
- Replace these with a set, having the combined probabilities of both elements;
- Repeat until only one element remains;
- This set is then split into a binary tree, which is used to encode the data.

### Activity 5: Using GIF Files

Experiment with reducing the quality and size of GIF files by changing colour palettes and introducing dither. Use whatever graphics applications you have, or go on the Web and use the online tools available there. Among others you might use Spinwave [[http:// www.spinwave.com/](http://www.spinwave.com/)].

Sites like these will invite you to select a GIF file on your computer hard disk or to select a GIF via an URL. The uploaded GIF is then compressed using their tool. All you have to do then is wait for the results! If you want a GIF on your disk for experimentation, save any GIF file from the many available on the Internet, or you can save and use this image below.



See Discussion on Activity 5 at the end of this chapter

### Activity 6: Image Sizes

Using the supplied graphics application, save different image types (photographs, line drawings, text) in different formats. Compare the file sizes of these formats, and experiment with different compression ratios, number of colours, and so on. Go to the Netscape homepage to examine the range of JPEG sizes.

See Discussion on Activity 6 at the end of this chapter

### Review

Check your progress by doing Review Questions 9, 10, 11 and 12

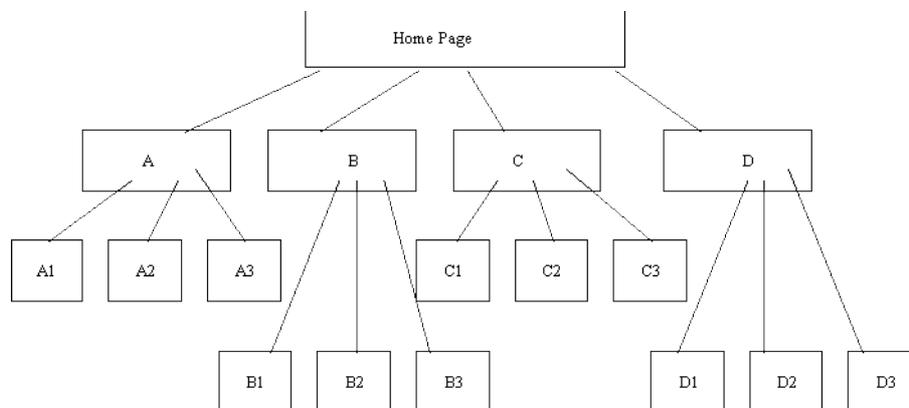
## 9.5 Designing Website File Structure

The final aspect of design Web developers have to deal with is the organisation of the files making up the website. This is particularly important if, as is typical, the website is to be developed by multiple individuals, and maintained until some future date. Web designers need to be familiar with the directory structure that a Web server assumes, and what the URLs mean in these terms.

### To Do

Read up on Web servers' directory structures in your Web design book, and then go on to read about what an URL means with respect to these structures.

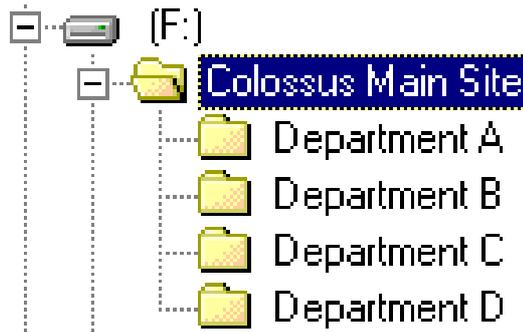
Recall the abstract information architecture of a website, which we explored using the example of a company that sells software development services and programming language implementations:



How would you implement this structure on a Web server? The most obvious way might be to locate the files related to individual pages in their own directories — one directory for the home page components, one for pages A1, A2, A3, B1, and so on. However, this may be excessive and lead to relative addressing mistakes. More importantly, such a design does not anticipate change. If the information architecture reflects departments of a business — department A, department B, etc. — then changes within the departments could cause a large

## Website Design

maintenance overhead. A preferable structure might have directories for the main departments only:



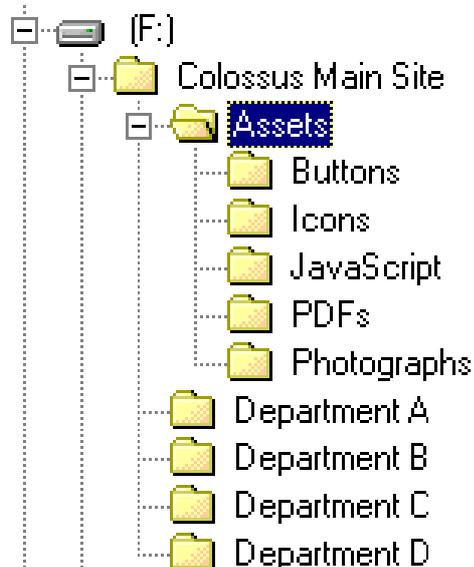
The folder for **Department A** would contain all the files for A1, A2 and A3 — both the HTML and image files (and maybe even the JavaScript files, QuickTime files, etc.). With this structure a link from A1 to A3 might occur as follows:

```
<A HREF="A3.HTM#pricelist">See price list for chairs</A>
```

On the other hand, a link from A1 to C2 would have to include a relative file path, as in:

```
<A HREF=" ../DepartmentD/C2.HTM#configurations">See computer configurations</A>
```

In fact, you will discover as you browse websites and develop your own, that it is useful to have a separate set of folders for generally useful components. For example, an Assets folder could contain files for GIF buttons, GIF or JPEG icons, JavaScript programmes, Acrobat PDF documents, and JPEG photographs:



## 9.6 Review Questions

### 9.6.1 Review Question 1

Name the aspects of website design.

Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.2 Review Question 2

What are the six main activities in site design?

Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.3 Review Question 3

What is the difference between an information architecture and a page architecture? Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.4 Review Question 4

Write down guidelines for navigation aids.

Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.5 Review Question 5

How does clarity affect the performance of users reading the content of a site? Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.6 Review Question 6

What aspects of download performance can a website designer not affect? Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.7 Review Question 7

Why is it important to specify the height and width of an image?

Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.8 Review Question 8

Give an example of a tag that would assist a search engine to find a website which sells software development consultancy services and programming language implementations (such as compilers or programming environments).

Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.9 Review Question 9

Name the three standard graphic image formats used on the Web, and note which is not yet widely used.

Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.10 Review Question 10

When should you use GIF formats?

Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.11 Review Question 11

Write down three advantages of GIF format. Answers can be found at the end of the Chapter in Discussions and Answers.

## 9.6.12 Review Question 12

What is a palette? Answers can be found at the end of the Chapter in Discussions and Answers.

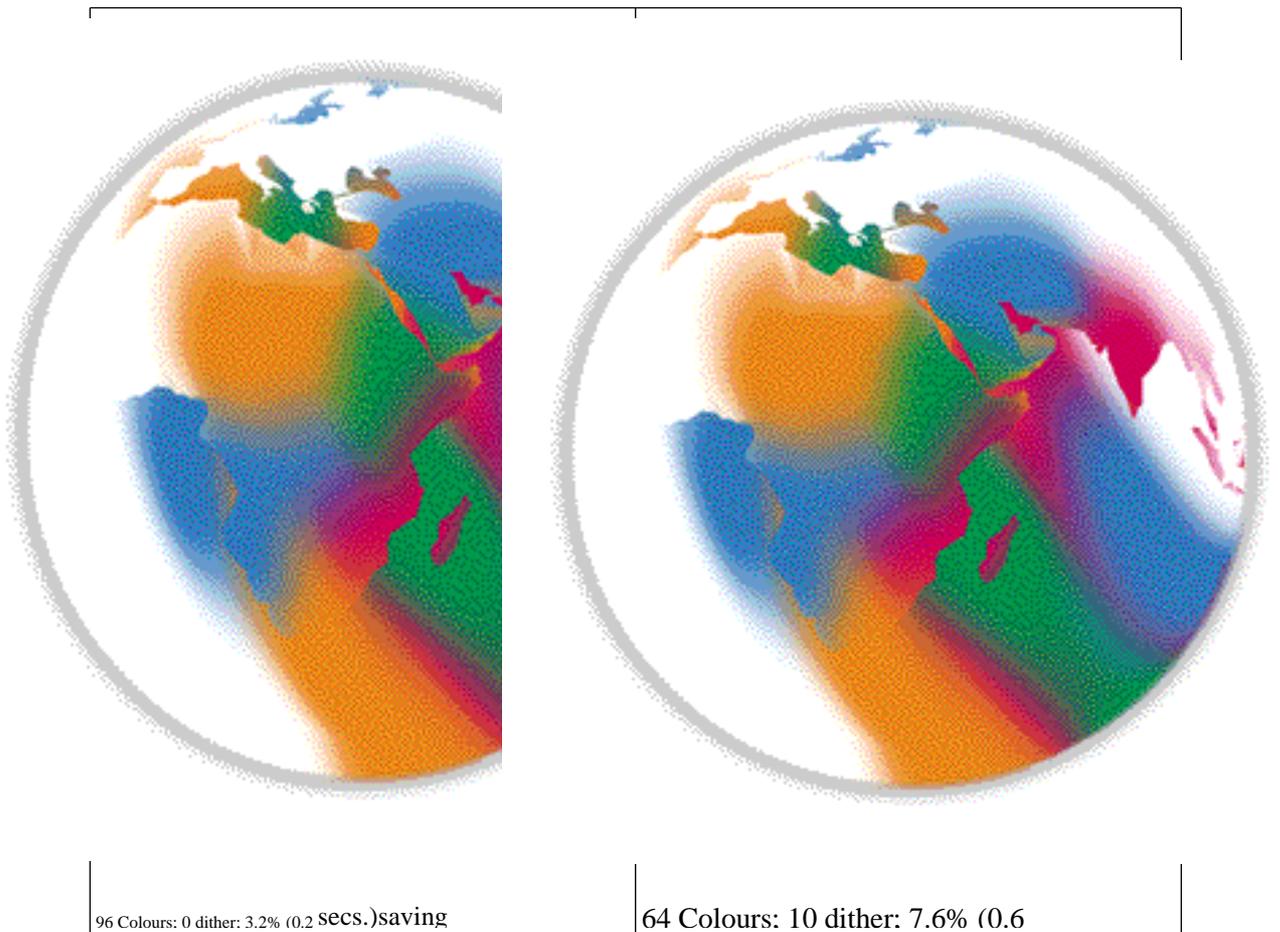
# 9.7 Discussion and Answers

## 9.7.1 Discussion of Activity 1: Bad Websites

The given two sites on poor Web design make for interesting exploration. Their designers have adapted the old adage of learning from other people's mistakes: 'learn good Web design from bad Web design'. Here we simply note common problems. The frame examples with bad navigation and overuse of JavaScript are clear examples of bad Web design.

## 9.7.2 Discussion of Activity 5: Using GIF Files

The following GIF images were produced by GIF Cruncher. The information given for each image are the tool's output. As you can see, you can gain dramatic space and time savings.





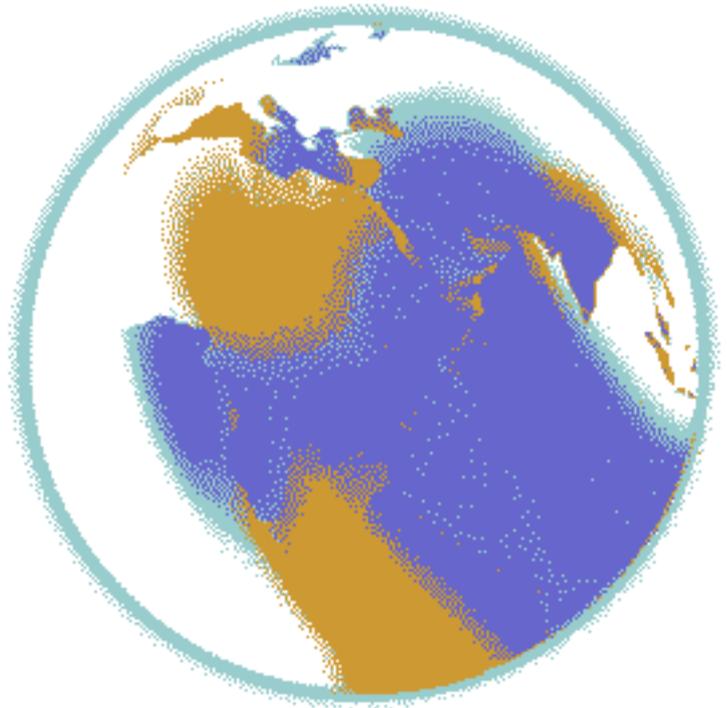
32 Colours; 20 dither; 26.3% (2.2 secs.)saving



16 Colours; 30 dither; 41.6% (3.4 secs.)saving



8 Colours; 40 dither; 51.7% (4.7 secs.)saving



8 Colours; 50 dither; 75.6% (6.3 secs.)saving



2 Colours; 60 dither; 85.7% (7.1 secs.)saving



2 Colours; 80 dither; 84.7% (7.0 secs.)saving

### 9.7.3 Discussion of Activity 6: Image Sizes

The Netscape site clearly shows how photographic images visually appear to suffer little from the quality degradation caused by 'lossy' compression.

You can experiment with JPEG files yourself using a graphics application or a website that offers the facility, such as JPEGCruncher [<http://www.spinwave.com/crunchers.html>]

### 9.7.4 Answer to Review Question 1

- choosing the user interface for the content you might wish to provide;
- choosing any graphic images you might need, and the appropriate technology;
- devising the directory and file structures that will allow you develop and maintain a site.

### 9.7.5 Answer to Review Question 2

- Analysing Overall Site Aims.
- website Architecture.
- Navigation Planning.
- Designing for Hyper-Reading.
- Estimating Download Times.
- Site Promotion.

### **9.7.6 Answer to Review Question 3**

An information architecture is a description of the structure of a site. Page architecture is a description of the structure of a page.

### **9.7.7 Answer to Review Question 4**

Every page on a site should include navigation facilities that allow the user to see where they have been (history), where they are (current), and where they can go (future).

### **9.7.8 Answer to Review Question 5**

Poor clarity in a website's content appears to slow readers down. This is probably because readers pause while trying to understand the content. Further, if users do not trust what they read they will question the usefulness of the site and may abandon it.

### **9.7.9 Answer to Review Question 6**

A website designer cannot affect the capabilities of a user's network connection, computer system (hardware and operating system) or browser.

### **9.7.10 Answer to Review Question 7**

By specifying the height and width of an image you allow a browser to continue to layout and render text while the image continues to download.

### **9.7.11 Answer to Review Question 8**

By specifying the height and width of an image you allow a browser to continue to layout and render text while the image continues to download.

### **9.7.12 Answer to Review Question 9**

GIF, JPEG, and PNG formats. PNG is not yet widely supported.

### **9.7.13 Answer to Review Question 10**

Answer: for two dimensional images, i.e. no lighting or shading required, compression will occur cleanly and precisely.

### **9.7.14 Answer to Review Question 11**

'Lossless' compression (LZW), transparency and interlacing.

### **9.7.15 Answer to Review Question 12**

A palette is a table of the colours available in an image. These colours are represent by indexes into the table.

# Chapter 10. Internet Commerce

## Table of Contents

Objectives.....	1
10.1 Advertising.....	1
10.1.1 The DAGMAR Strategy.....	1
10.1.2 The Advertising Plan.....	3
10.1.3 Advantages to Advertising on the Internet.....	4
10.1.4 Disadvantages to Advertising on the Internet.....	5
10.1.5 Advertising to support your site.....	5
10.1.6 Other ways to find 'the' website.....	6
10.1.7 Getting Visitors.....	7
10.1.8 Advertising Review.....	7
10.2 Selling.....	8
10.2.1 Everyone's doing it.....	8
10.2.2 Selling is Similar to Advertising.....	9
10.2.3 Wholesalers and Retailers.....	9
10.2.4 Security and Selling.....	9
10.2.5 Usability for Internet commerce sites.....	10
10.2.6 The Virtual Shopping Trolley.....	11
10.2.7 Delivery.....	11
10.2.8 Selling Review.....	11
10.3 Review Questions.....	12
10.4 Discussions and Answers.....	12
10.4.1 Discussion of Exercise 1.....	13
10.4.2 Discussion of Exercise 2.....	13
10.4.3 Discussion of Exercise 3.....	14
10.4.4 Review Question 1.....	14
10.4.5 Review Question 2.....	14
10.4.6 Review Question 3.....	14
10.4.7 Review Question 4.....	14
10.4.8 Review Question 5.....	15
10.4.9 Review Question 6.....	15
10.4.10 Review Question 7.....	15
10.4.11 Review Question 8.....	15
10.4.12 Review Question 9.....	16
10.4.13 Review Question 10.....	16

## Objectives

At the end of this chapter you will understand:

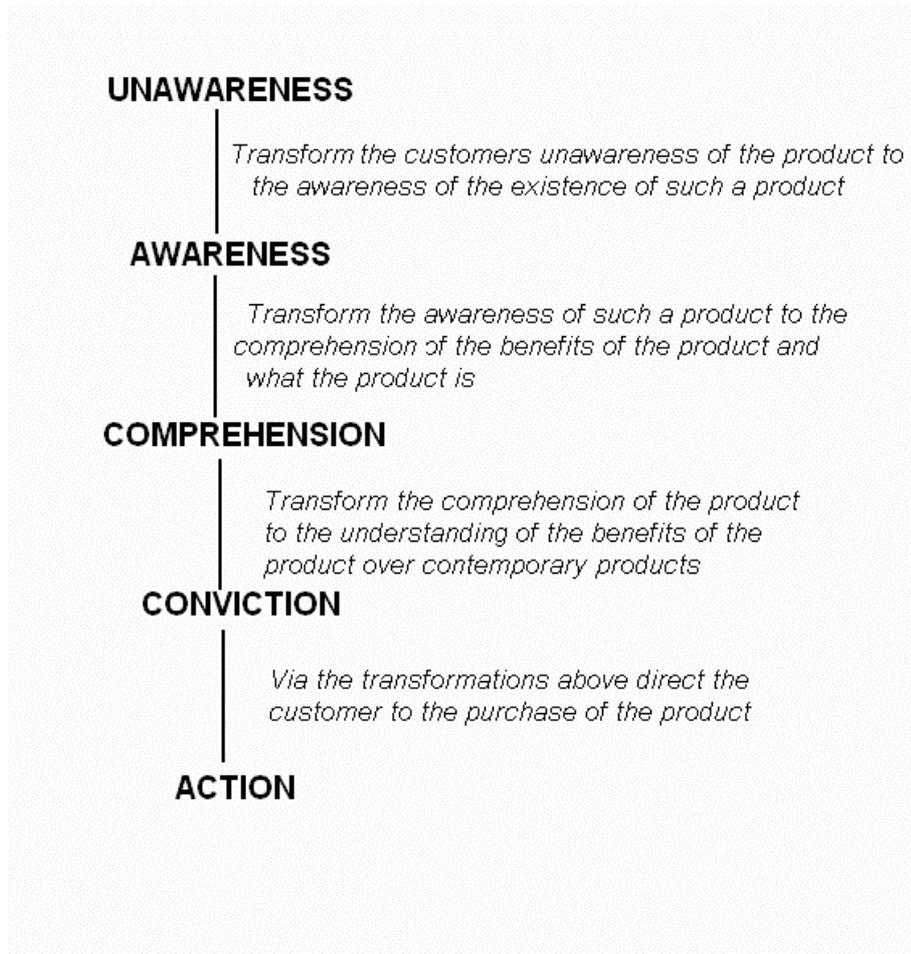
- How to advertise your website; and
- How to sell goods through an ecommerce website.

## 10.1 Advertising

### 10.1.1 The DAGMAR Strategy

Advertising is a subsection of marketing, and marketing is a company's ability to profitably manage its customer base.

The advent of the Internet is causing the redevelopment of marketing strategies in all business. This unit covers an existing advertising strategy: DAGMAR — Defining Advertising Goals for Measured Advertising Results. This strategy attempts to move a customer from a state 'unawareness' concerning the product or service, to a state of 'action', in which it is hoped that the customer will buy the product. The four stages of DAGMAR are illustrated below:



### **Awareness**

A major marketing concern facing the Internet is attracting customers to a website. In traditional media, such as the press and television, an audience is researched and targeted accordingly. Targeting via the Internet is difficult, but alternatives exist. For example, raising a site's search profile on the popular search engines such as Yahoo and Google, or by providing links to your site from Web pages most likely to be visited regularly by your target audience.

### **Comprehension**

Once the target audience has been attracted, the website must explain, in a concise way, what the characteristics differentiating its featured product from that of its competitors are. Pictures might be used, as well as a list of its technical features and benefits.

### **Conviction**

Once the target audience has been attracted, the website must explain, in a concise way, what the characteristics differentiating its featured product from that of its competitors are. Pictures might be used, as well as a list of its technical features and benefits.

### **Action**

Making claims about the product is, however, not sufficient. The audience must also be convinced that the claims are genuine. For instance, a lifetime guarantee could be given, or the claims could be backed with specific

evidence.

The action stage does not necessarily involve a purchase. However, the customer may still want further information concerning the product, or even demonstration (this is very popular with software). This is often where failure occurs, due to difficulties in communication via the Internet.

## **To Do**

Find out more about cybermarketing in your textbooks and on the Internet.

## **10.1.2 The Advertising Plan**

### **What does the customer need?**

You need to ensure that the website has sufficient information for your potential customer to make an informed decision as to whether or not your advertised product or service is of interest. Over and above supplying information, the website should also persuade the customer that the product or company is indeed of interest.

### **How large and complex does your website need to be?**

Much effort and money can be invested in developing a large and complex website to advertise any particular product. However, if the product is not profitable, then those resources have been wasted. Alternatively, very little can be invested in developing a simple website. This runs the risk of offering smaller returns than a larger website would offer. Clearly, the cost for developing an appropriate website needs to be decided.

### **Is your product conducive to the Web?**

Software sells well on the Web, apples and oranges less so. However, the Web has a great deal of flexibility: it might be difficult to sell films on the Web, but a site listing theatre performances for the local theatre can be very useful, and might lead to more business.

### **Do you want to hire someone?**

Various experts sell their Web expertise, and they may be able to help with the design, implementation or maintenance of a website. Do you want to hire them, or do you want to develop the site yourself?

While a simple website requires little expertise, hiring an external company or independent contractor is probably beneficial for any medium-sized website. A large website may require employing new staff. In this case, all the work may be kept internal, or another company could be hired.

## **To Do**

Go to the Interactive Advertising Bureau website [<http://www.iab.net>]. Here you will find a variety of interesting articles on Internet advertising.

### **What information should appear in an advert?**

There are many levels of advertising. A small company's Web site is likely less sophisticated and, for that matter, smaller than a large company's (such as General Motor's Web site — the URL is included at the bottom of the Contents section).

Different forms of information require different levels of sophistication in order to display it. Below is a list of information that could be provided on a website, ordered from 'least sophisticated' to 'most sophisticated'.

### **Contact information**

Contact information should be provided on any site. Supplying contact information allows a visitor to easily obtain more details concerning a product or service, if they so wish. Contact information should include the address of the company, the telephone number, and the name of the person to contact if appropriate. It may also include an email address.

### **Product descriptions**

The 'product' being advertised may be a specific item, such as a watch, or it may be a service, such as bookkeeping. Initial product descriptions should be limited, but as the site grows larger (assuming that it is successful), more product information should be included.

### **More sophisticated information**

Further information can come in many forms: product specifications, testimonials, pictures and graphics showing the product or company offices, and so on. A site for the local library might include information about library events or stock additions.

### **Links to related information**

Providing links to more information may encourage visitors to buy the product or service. This is a, essentially, a trust building exercise — if the site has something good to advertise, providing more information about can only increase the visitors' trust.

### **Information on Selling**

Advertising a product is not the same as selling a product. A website can attempt to do both (see the next Section), but should make it clear in the advertising portion of the site that on-line sales are available.

## **10.1.3 Advantages to Advertising on the Internet**

### **A big audience**

Many people use the Internet every day, but making them visit a particular website can be difficult. Appropriate visitors can more easily be found — and the site's hit rate hence improved — by using an HTML document's META tag.

### **24/7**

The Web is available 24 hours a day and seven days a week. It is available all around the world. Every browser accessing the Web is a potential visitor and customer, and customers can visit a site at any time they find convenient, day or night.

### **An interested audience**

An advertising billboard attracts the attention of many people, even those not interested in the product. It is seen only by those people in a particular area. People use the Internet to look for a particular product. If the product happens to be a product you are attempting to sell via a website, and the site can easily be found, it is much easier for your potential customers to find out about the product than via a billboard. A billboard may be said to be creating demand — the website is not creating demand, but instead making shopping easier for the customer.

### **Possibly quite inexpensive**

The connection fees, hardware and software costs for setting up and running a website are minimal. Using an ISP (Internet Service Provider), running a simple website can cost under R100 per month. A Web server can be run on a standard PC, or, alternatively, free or commercial Web-hosting service can be used. As a result, many people have their own websites.

### **You can collect customer information**

Collecting visitor data can generate useful information concerning customer interests and preferences. Collection can be done in a number of ways, for example: visitor actions can be recorded; visitors can be asked to fill in forms supplying information; their email addresses and interests can be recorded. Once this data has been collected, regular updates can be sent to customers using either email or physical mail, informing them of new products and further information.

It is important to keep customers happy, as well as to keep their interest on the website — sending a lot of unwanted mail, however, can damage this customer relationship.

## To Do

- Visit Search Engine Watch's tip page [<http://searchenginewatch.com/webmasters>] for submitting to search engines.
- Read up about how businesses make use of the Internet to market their products and services.

## 10.1.4 Disadvantages to Advertising on the Internet

### Being unable to show the product

When advertising software on a Web-site, the customer is always able to download and run a demonstration copy of the software. On the other hand, it is difficult to demonstrate the freshness and ripeness of any fruit or flowers, for example that might be sold via a website. Showing a photograph of a mango does not let a customer examine the fruit in any way. Similarly, photographs of an automobile and a list of its specifications are informative, but customers are unable to drive the car. This problem can be lessened by supplying quality guarantees. Remember that the main goal of advertising is to catch the attention of possible customers, and hopefully a well-designed website will encourage them to seek further information, and possibly purchase the product or service.

### Emotions cannot be easily communicated

The means to communicate emotions through non-verbal contact are limited, and therefore certain emotions that would be detected in person-to-person conversation are not easily detected in email or other textual information exchange. For instance, it is difficult to tell via email if a customer really is delighted to be investing a product.

### The global user cannot always physically reach a business

When running a local business — a bakery, for instance — the benefits of advertising via the Web on the global market are lost. If the business is based in London, it does not matter if someone in New York City can see its website, because they cannot reach the shop to buy any goods.

### A security breach can really hurt

If someone gains enough access to your site in order to edit its content, they will be able to negatively effect its advertising. For instance, Governmental sites have been broken into, and the Web pages have been changed to display pornography.

### Competitors can see the site

The competition can look at your site, see the listed prices, the available stock, and so on. Hence, a website provides competitors with easy intelligence. However, this is information that could probably be found elsewhere by any interested party, and so is not a major disadvantage.

### Review Question

Do Review Questions 1, 2, 3 and 4.

## 10.1.5 Advertising to support your site

It is important for a website itself to be advertised. Therefore, when building an informational website (as opposed to a commercial site), it is possible find funding by advertising other sites on your own.

Many larger websites have a large income from advertising other sites. This can be irritating to visitors, however, as they can easily be bombarded with a lot of (mostly) useless information.

Some ISPs, such as Geocities, provide free Web access. They can do this because, among other reasons, they advertise other products. This means that when one of the ISP's customers is browsing the Web they also see

advertising placed there by that particular ISP.

An informational site about a particular area of interest often has people wanting the site to link to their sites, especially when they are selling a product related to the informational site's area of interest. These people could be charged in order to have such a link.

One factor to consider when advertising someone else's site is the quality of their product or service. You might want to only advertise what you consider to be good quality products. Showing their advertisement could be thought of as offering your seal of approval on their product.

## 10.1.6 Other ways to find 'the' website

People search the Internet for websites that interest them. The designer's job is to make it easy for them to find your site. The big search engines can easily have people visit at a site, provided the META tags are used appropriately. Also, linking to the site in as many places as possible and appropriate helps as well. There are a number of websites that act as central reference sources; these often link to other websites. It is important to have your site linked from all such appropriate centres. Some of these sites are:

- The **product** website: When selling a product, it is very likely that there is already a general website for the kind of product you are selling. For example, when selling printers there is likely to be a general website for printers. Such a site may link to other sites that have information about printers, as well as to sites that sell printers. If there is such a site for your product, ensure that your site is listed here.

Some people predict that in the near future product Web sites will abound; they will be how people search the Internet.

- The **area** website: If your website advertises a service or shop based in a particular location, then it is worth finding websites for the area as a whole. If such a site exists, then you want to be listed on it. Such area sites often link to local shops and amenities. For example, if your website is for a theatre in Islington, London, then your site should be linked to on Islington's website.
- The **interest group's** Web site: A website that supports a group of people who share similar interests is an ideal place to link to your Web site, provided that you can identify an interest group matching your target audience. For instance, when building a site to advertise a series of painting exhibitions, link to the site from the home pages of local artist's.

Advertising another site is one way to fund your site.

### Activity 1: Checking your site popularity

You can determine your site's popularity in many ways. We are going to illustrate these methods by using the the University of Cape Town's website [<http://www.uct.ac.za>].

The first method is to employ a ranking tool such as the one at Mike's Marketing Tools [<http://www.mikes-marketing-tools.com/ranking-reports/>]. Such a tool can be used to rank a site with respect to particular keywords. In the case of UCT, we can find its ranking with respect to the key word (or phrase) 'Cape Town': in other words, this is how high the UCT site ranks when people enter the phrase 'Cape Town' in a search engine. Of course, you could manually enter the phrase in all the search engines yourself, but these tools automate this process. Follow these steps:

1. Go to the Mike's Marketing Tools — Search Engine Rankings Tool [<http://www.mikes-marketing-tools.com/ranking-reports/>].
2. Look for the 'Submission Form' section
3. In the URL enter [www.uct.ac.za](http://www.uct.ac.za)
4. In the keyword enter Cape Town
5. Submit the request by pressing the 'Check Rankings' button
6. Analyse the results and reflect on what they mean for the UCT website. Try key words and sites.

Another method is to determine the link popularity for the website. Link popularity refers to how often other

pages link to the site's address. You can do this by using a free tool such as 'http://www.webseoanalytics.com/free/seo-tools/link-popularity-checker.php'. Enter 'www.uct.ac.za' in the url search back and see what results are returned.

## 10.1.7 Getting Visitors

No matter how good a site is, it is useless unless people visit it. Here are some useful tips for attracting visitors to a site:

- Add it to a "What's New" Web Page. Some sites maintain lists of new websites, in essence advertising these sites. These "What's New" pages tend to be viewed mostly by the more sophisticated user, which makes it less than ideal for attracting new customers.
- Submit the site to a Web Directory, such as Yahoo [http://www.yahoo.com] or DMoz [www.dmoz.com]. Web directories are designed to help users find the sites they are looking for by browsing — rather than searching — the Internet.
- Make the page searchable. Search engines automatically search documents and classify them, looking for information in the title, the top of the page, and in the HTML META tags. Pages can be made searchable by giving it a good title, putting appropriate keywords near the top of the page, and making use of the META tags.
- Announce the site using Newsgroups, Newsletters, Books, and Magazines advertising sites.
- Use lists of lists. These applications reference a site from hundreds of other sites. In many cases, these sites require sites that they link to reciprocate, by advertising them in turn.

### Activity 2: META Tags

Below is an example of a typical title and meta tag applied to a company, Cape Biscuits. The keywords used are biscuits, cookies and savouries.

```
<TITLE>Cape Biscuits CC.<TITLE>  
<META name="keywords" content="biscuits, savouries, cookies">
```

Design a website for a company that produces Crisps. The name of the company is, "The Crisp Factory". Choose a suitable title and keywords for the opening Web page.

## 10.1.8 Advertising Review

Advertising is a sub-field of marketing. At the beginning of the section we examined a strategy based on the DAGMAR model, but we have not yet discussed this strategy in any detail with respect to the Internet. Below we return to three of the four transformations in the DAGMAR model. The fourth transformation, action, is left to the next section.

### Awareness

Product awareness can be improved by submitting a Web page to one of the major search engines, or by linking to the site from other Web pages popular with your target audience. Insert appropriate keywords in the META and TITLE HTML tags. Carefully chosen keywords can greatly enhance the likelihood of a page being identified by a particular search engine. One important method in keyword selection is to use synonyms. Synonyms can be found by performing a search with a given keyword and finding associated topics.

Awareness of a website can also be improved by linking from other popular websites, especially those with a similar topic area.

### Comprehension

Product comprehension can be improved by providing detailed product information, statistics and / or scientific results. It is important, however, to allow the customer to choose how much information that they receive. It is

possible to lose customers by inundating them with unwanted information.

## Conviction

Winning customer confidence can usually be best achieved by giving away samples. This often results in an 'action' (in the DAGMAR sense) from the customer's point of view, and might mean that the customer might never make it to the action of buying the product! There is a thin line between these two transformations: from Comprehension to Conviction, and from Conviction to Action. With software advertising, for example, demo software is often made available. This is good from the customer's point of view, since the demo gives them a much better understanding of the final product. This increases their confidence in the product. It is also healthy from the company's perspective as it allows them to receive important information about their customers, such as the networking domain they are attached to, employment and general demographics.

### Exercise 1: Walt's Music Shop 1

Walt has several Music shops from which sells both new and used CDs and LPs. He began his career 20 years ago selling used LPs out of a small second floor room. After years of only scraping by and working very hard Walt expanded to three shops, all of which are in the same large metropolitan area (which we will call Metropolis).

The MP3 hype on the Web has made Walt interested in computers. He's bought a computer, and he and a few of his employees are now avid Web surfers.

Walt wants to do some advertising on the Web. He's thought about selling items on-line, but has decided, at least for now, only to advertise. Do you think Walt is wise to advertise on the Web? Write approximately 150 words justifying your answer and suggesting how Walt could use his site to support his business. Write your response as if you were addressing Walt himself.

Read Discussion of Exercise 1 at the end of the Unit.

### Exercise 2: Walt's Music Shop 2

Walt is convinced by your answer. Realising that you are something of an expert in this area, he asks what you think he should put on his site.

Walt does not have much of an advertising budget and is unsure how much this site would cost. Provide him with three options: one for an inexpensive site, another for a slightly more expensive site, and finally one for a full-blown, highly interactive website. Which one would you recommend?

Read Discussion of Exercise 2 at the end of the Unit.

## 10.2 Selling

### 10.2.1 Everyone's doing it

Many people sell things via the Web. Statistics — always of varying quality — concerning how many people are buying things, selling things, and how much money is being transacted, abound. There are predictions of the future value of on-line sales stretching into trillions of dollars. Some predictions even claim that the majority of items bought and sold in the world will eventually be bought and sold over the Internet.

Even with a healthy scepticism of statistics and predictions, it is still clear that there are many Internet sales currently taking place, and that this number seems likely to only increase.

This section explores the following questions:

- How are things sold using this medium?
- What should websites for Internet commerce look like to perform successful sales?

## 10.2.2 Selling is Similar to Advertising

Selling on the Internet has largely the same advantages and disadvantages as advertising on the Internet.

### Advantages

- A Big Audience
- 24/7
- An interested audience. Just because a website appears popular does not mean that it truly is, or even that it's useful. For example, a high hit rate could simply mean that a lot of people visited the page by mistake.
- Possibly quite inexpensive: running on-line sales is more expensive than advertising alone, but should still be relatively inexpensive.
- You can gather customer information

### Disadvantages

- You cannot show the product, although quality guarantees can be used to counteract this.
- Your competition sees the site
- Security breaches are dangerous. Firstly, computer criminals are highly motivated to break into on-line systems when money is involved. Secondly, if customers are defrauded, future sales from your site may be effected.
- A possible disadvantage of advertising is that global users cannot always get to you. Sales have a similar problem: the product must somehow reach the customer.
- Nothing can be sold unless people visit the site. This means that the site has to be advertised.

## 10.2.3 Wholesalers and Retailers

Wholesalers buy a particular product in bulk and sell it on to retailers, who deal directly with the customer. Internet sales break down this relationship. This can have both positive and negative effects on the industry concerned. One positive consequence of introducing Internet sales is that it cuts out the middlemen (i.e. the wholesalers and retailers), and therefore the product price can be reduced. However, eradicating wholesalers can be a risk to the supplier. Since wholesalers buy the product in bulk, they pay for the space to store your product, and also take the risk of buying the product without knowing its shelf-life. Retailers deliver the product to the customer and handle the majority of customer complaints. Selling directly to the customers removes these middlemen, so successful Internet sales require the implementation of the following:

- increased storage space
- tighter predictions concerning the demand for the product
- a good delivery system to the customer
- increased staff for customer relations

### To Do

Find out more about how Internet commerce has affected the man on the street. Visit websites such as EBay [<http://www.ebay.com>] and Amazon [<http://www.amazon.com>]. List the reasons why you think people would or would not buy or sell items using these services.

## 10.2.4 Security and Selling

When selling something over the Internet, money has to be transferred between people. Money transfer is a potential target for misuse and is therefore a possible security problem. However, this problem has been studied for many years and there are now a number of technologies, largely encryption-based, that exist to secure communication channels. Under this approach, information about the purchase, including the user's credit card details, is encrypted in such a way that the seller is the only one who can decrypt it. Hence, the transaction is secure.

Examples of these technologies include cyber-cash, digital signatures and PGP encoding of the transaction.

Of course, the provider (the person building and supporting the site) must provide the security features. These may be transparent to the user (i.e. the user is not required to do anything special) but they must exist.

A major concern to customers is that their details are adequately protected. Customers should be provided with assurances that their transactions are indeed secure, and there should be explanations of how this security is ensured.

Security complicates matters, but adequate precautions can be taken. Currently a customer is more likely to be defrauded by a waiter in a restaurant via a credit card transaction than by someone gaining access to a secure communication channel used for e-commerce.

For a more thorough discussion of security see Unit 13.

## Review Questions

Do Review Questions 7—8.

## 10.2.5 Usability for Internet commerce sites

High level usability considerations such as consistency, adequate feedback, good use of screen geography, colour and sound, navigability, on-line help, and so on, are as important for a website as it is for other interactive systems. Indeed, some would argue that they are even more important.

Sites should be easily understood by a first-time user. It is important to realise that if a first-time user does not find the site usable, it is unlikely that they will return.

A site needs to guide the visitor through the transformations in the DAGMAR model in an easy and confident manner. This means that a visitor should be able to browse a collection of products on offer, browse the products' details, receive a demonstration of any particular product, purchase a product, and feel confident that the transaction has been performed correctly and safely. Usability is a key consideration at every step. For example, if it is not clear when the purchase has been accepted, not only is it frustrating for the user, it might result in multiple purchases being inadvertently made — or, indeed, no further purchases at all.

One important user task is the ability to browse a product list, perhaps to confirm product details or access further product information. Whatever the reason, the implementation of this task should be well considered, especially how the product list can best be displayed, ordered, and searched. Simply listing the products in numerical order will not support even a simple product search: users likely do not know the product's position in the list, and it is incorrect to assume that users know exactly which product they wish to buy when they first visit the site.

While usability is important to an advertising site, it is even more essential at a point of sale — this is the only chance customers have to confirm that they want a product before they buy it.

Providing a mechanism to enquire about the products on offer is a minimum user requirement. This can be achieved by, for example, providing the e-mail details of a virtual shop attendant. This is, essentially, one type of on-line help system.

Repeat visitors / customers should also be considered. It is possible to ignore them, but if customers are expected to repeatedly purchase products it becomes good customer relations (and good usability) to provide website functionality targeted towards them. This could be done by keeping an individual record of a customer's purchases, including how often they have visited the site. If a high percentage of a customer's recent visits have resulted in the purchase of the same or similar goods, it might be appropriate to offer them these goods on their next visit, or to suggest similar products. A record of customer details, such as address and payment details, could be stored, which stops customers from having to repeatedly supply these details for every purchase.

Remember that Internet commerce sites are expected to be highly interactive, and should provide high levels of support for the customer.

## 10.2.6 The Virtual Shopping Trolley

The details of how a user browses, inspects, selects, and purchases items differs from site to site. However, a common metaphor has been developed: that of the 'virtual shopping trolley', or 'virtual shopping cart'.

The metaphor can be thought of as going to a supermarket: items are taken from various shelves and placed in a shopping trolley. The customer carries the items with them as they move through the store, and are free to add more things to the cart or even to return items to the shelves. Once the customer has selected all the items that they want, the trolley is taken to a cash register and the goods are paid for.

Using this metaphor for on-line shopping is usually a good thing, as lots of people are already familiar with it from their own lives. A virtual shopping trolley operates in much the same way as a real trolley: items can be selected from those available on the site and placed in a virtual trolley. Once in the trolley they can be examined further or returned to the store. Finally, once the customer has finished selecting all of their goods, all the goods in the trolley are paid for and shipped to the customer.

While this is the basic shopping trolley metaphor, further functionality can be added to it. For instance, a website can, for the customers' convenience, keep a running total of the cost of the items in the trolley.

## 10.2.7 Delivery

Once the purchase has been completed, the products must be delivered. Sometimes this is trivial: if the 'product' is a hotel reservation, confirmation can be sent by email and nothing physical need be delivered. Software can also be delivered over the Internet (through email, or FTP, for instance), although if the software comes with paper-based support materials, such as a manual, then physical delivery is still an issue. Most items need to be physically delivered. The problem of physical delivery is also faced by telephone and mail orders, and so is nothing new.

It is important to remember that since the Internet gives access to a global market, a company's existing delivery channels may not be adequate for their website's potential customer base. One common form of delivery is Express Mail, which allows books, music, documents, and other small items to reach anywhere in the world for a reasonable price. However, alternative possibilities will be required for some items: for instance, a store may not be able to deliver fresh flowers through standard Express Mail, but it may be possible to fulfil orders using a store close to the customer.

Useful functionality for an e-commerce site is to allow customers to check on the progress of their delivery, such as by finding where their order is in the delivery chain, and how long they will have to wait for it to arrive.

## Review Questions

Do Review Questions 9—10.

## 10.2.8 Selling Review

The final customer transformation in the DAGMAR model is action. The action, as mentioned above in the advertising review, is the customer ultimately purchasing an item from the site. Only a thin line separates this stage from the conviction stage. Usually, once the customer is convinced, an action is immediate. This action may take the form of either trying a product sample, or buying the product. From the seller's point of view, of course, the purchase action is the more desirable action.

Selling is similar to advertising, but Internet sales raise two further issues:

- Security. Sales require a money transfer, and customers must be confident that the transaction can be accomplished smoothly, accurately, and securely, otherwise no sale will result
- Delivery. The Internet gives access to a global market, requiring product delivery to anywhere in the world. This requires appropriate delivery channels to be established.

Site usability is of paramount importance. Customers deserve, and will demand, high levels of support and easy access. Remember that customers can easily go elsewhere if a site is confusing or difficult to use. Issues concerned with user interface design were dealt with in Unit 3.

### **Exercise 3: Walt's Music Shop 3**

Consider Walt's Music Shop from a previous exercise. Walt now wants to try to sell records over the Web. What does Walt need to do?

Read Discussion of Exercise 3 at the end of the Unit.

## **10.3 Review Questions**

1. When designing a website to advertise a product, what design issues should be considered other than aesthetic and usability ones?

Answer at the end of the chapter

2. Imagine that you are constructing a website to advertise a local coach or bus company. What information would you include on the site?

Answer at the end of the chapter

3. Your employer wishes to build a website to advertise the company, but is apprehensive and needs some reassurance. Write a short essay (no more than 300 words) explaining the advantages of marketing on the Internet.

Answer at the end of the chapter

4. List the main disadvantages of advertising on the Internet. Answer at the end of the chapter

5. What are the financial advantages of having a popular Web site? Answer at the end of the chapter

6. Why is it necessary to increase awareness of the product or service you offer among your customers, and what techniques can you use on the Internet to increase their awareness?

Answer at the end of the chapter

7. Explain some of the advantages and disadvantages of selling products on the Internet. Answer at the end of the chapter

8. Explain how an organisation may restructure itself once it begins advertising on the Internet. Answer at the end of the chapter

9. Explain improvements that can be made to a website to cater for returning customers, as well as to the marketing of several products (for instance, as in with a virtual bookshop such as Amazon).

Answer at the end of the chapter

10. Refer to the DAGMAR model and explain how the different transformations of the customer's state may be achieved in the context of the Internet.

Answer at the end of the chapter

## **10.4 Discussions and Answers**

## 10.4.1 Discussion of Exercise 1

Your answer might be something along the line of:

*Yes Walt, I think you should put up a site advertising your shops. You could find a lot of interested customers. While it is unlikely that you'll get customers from around the World, it is likely that you will be able to reach thousands of customers here in Metropolis. You can provide support for people who already know about your shop, and you might even be able to attract a few new customers. You can probably put it on your ISP's server for under 100 dollars a month and maybe even for free. You're not really going to be giving away anything to the competition: he's in here every week anyway. Finally, it might help you to learn something more about your customers, especially those shy ones who never speak to you when they come in.*

*You should be aware that, like you, a lot of music enthusiasts are hitting the Web. Some people are actually distributing music this way. This kind of delivery might cut into your business, but if you pay attention you might be able to use the medium to enhance your business.*

Note that this is just an answer, not *the* answer. Other points are relevant. If you've come up with some different points, discuss them with your classmates.

## 10.4.2 Discussion of Exercise 2

Your answer might be something along the line of:

*Well Walt, let me propose three options:*

- 1. A simple yet effective site could include just a few pages, or even just one. Seeing as you're getting relatively handy with a computer, you might be able to do this yourself. This should have a list of the addresses of your stores, their phone numbers, the managers' names, and the hours that the stores are open. Of course you'd put in your email address. You might want to throw a bit in about your LP and CD collection, and maybe even advertise a few specials. You probably could put this together reasonably quickly for just the cost of your time.*
- 2. A slightly more complex site would include all of the information mentioned in the first option, but would also have some more sophisticated material. You'd probably need someone who has some skills in Web-based applications, but they could put it together in a day or so. Once developed, you wouldn't have to modify it, so maintenance costs would be negligible.*

*This version could include a form for visitors to supply some information about themselves. This might include the music they're interested in, where they live, and their email address. You might want to send out mailings to your customers for special events, so you could ask them if they'd like to receive this kind of information. (Don't just send them information as the spam could offend your customers and do more harm than good.) If you do this, you should also include a text box so that the users can write in anything that isn't included in your form. Other pages could include details about your stock.*

*If the site grows to be more than 10 pages, you'll need to consider navigation issues to make sure that the site is usable and that visitors don't get lost.*

- 3. You could go whole hog, and put your entire catalogue of records on the Internet. This would require a lot of maintenance, especially if the catalogue frequently changes. You could hook the website up to your stock-taking programme (you do have one of those, don't you? If not, you probably should invest in automating that). That'll take some serious coding, but this could be a real boon for your customers. They could search the collection from their home, and you might even provide a mechanism for them to reserve records, or to order records that you don't have in stock. If you do that you'll have some security issues, but they can be solved. This will probably also require some regular maintenance. You will need to recruit some extra help for this style of Web page. I wouldn't recommend that you try and produce this yourself.*

*Other neat things you could do include providing music samples, or perhaps links to other local music sites and live music venues. In fact, this might be useful to do for even the simple version of the site. It'll only bring more people to your site, even if they're on the way to elsewhere. The sites that you link to might be willing to link to you in return. You could even sell over the net.*

*I recommend that you go for the second option. It's not really that expensive, so you're not taking much of a risk. If it goes well you can always upgrade to the more sophisticated site later, or do so gradually, as you learn more about your customers' needs. Whichever option you choose, you will have to advertise the site to get people to come to visit it. If you want more ideas on that, just ask!*

### **10.4.3 Discussion of Exercise 3**

Walt needs to provide a way of showing the product and their prices to the user. This could simply be a list of album titles and artists, though more extensive information would be useful. For example, you might also include the album tracks, its release date, and all of the artists performing on it.

An interface for purchasing various items will be needed. Since it is likely that people will buy multiple items, a shopping trolley could be used. It would be really good to have a tutorial to help the first time user use the system if they needed it. For return customers, you might want to suggest products based on their previous purchases. This could be irritating to a customer, so there should be some way for them to turn this feature off.

### **10.4.4 Review Question 1**

You should consider the following:

- The size of the website
- How complex or how simple should the website be? Or does it need to be?
- Should you employ external contractors or employ new staff?
- What information should the website include about your product?

### **10.4.5 Review Question 2**

The advertisement should include the following items of information (the list is not exhaustive — it is only indicative).

- Individual routes and a timetable.
- Prices, including any special offers or cheap ticket deals. This may involve too much maintenance, but we don't have any information about budget constraints.
- Quality of service, such as the type of coach you use.
- Any guarantees offered regarding quality and punctuality of service. If statistics are available, such as percentage of coaches running on time for the last year, this would build customers' confidence in the offered service.
- How a customer can book a ticket. For example: contact details for the booking office, and opening hours.

### **10.4.6 Review Question 3**

The main advantages of advertising on the Internet are:

- It is open 24 hours, 7 days a week
- There is a very big potential audience.
- It is relatively inexpensive compared to, say, television advertisements.
- You can easily find out more about your customers via forms and questionnaires
- Sales via the Internet cut out the middleman, and hence reduce the product cost.

### **10.4.7 Review Question 4**

The main disadvantages are:

- It is difficult to give the customer a realistic product experience. For example, The Web is not ideal for selling fruits or other products with a short shelf life.
- Locality — a small Pizza Bar in Las Vegas, may be able to sell Bagels anywhere within Nevada via the Internet, however, it would be a stretch for a customer in Madrid to make an order.
- Security breaches can have an adverse effect on your company.
- Wholesalers/Retailers often order in bulk and therefore take away a large portion of your stock. If you are relying on sale from the Internet you may have to consider extra storage space for your advertised product.

## 10.4.8 Review Question 5

Advantages:

- Popular websites have a large number of hits (e.g. 1 - 2 million per day).
- Popular websites charge for advertisement space.
- The revenue created by advertisements can support the upkeep of the site e.g. see Yahoo.

## 10.4.9 Review Question 6

Awareness is necessary because the amount of customers visit the site, as well as their behaviour, directly relates to your profit. On the Internet, customers have to come to you, and this means that you need to lure your customers to you.

You can increase awareness of your website by advertising on other sites such as the major search engines or interest group pages, and by making your website attractive. Good websites have very high hit rates (up to 1 million per day, a very big potential audience).

## 10.4.10 Review Question 7

Advantages:

- It is open 24 hours a day, seven days a week.
- It never tires, and never shows the emotions of working 12 hour shift.
- You potentially have a huge audience (50 million).
- It is relatively inexpensive.
- You can gather information concerning your audience.
- The price of your product is reduced by cutting out the middleman.

Disadvantages:

- Product usually cannot be physically experienced or demonstrated.
- Your competition sees the site, and is aware of your product information and development.
- A security breach can have a detrimental effect on your customer base.
- Cutting out the middleman has implications concerning your organisational structure.

## 10.4.11 Review Question 8

Changes that may need to be made include:

- Increasing storage space for stock.
- Retraining of staff to handle the increased direct customer contact.
- Production will be based on a supply and demand strategy rather than mass production. The bulk of storage, and hence shelf-life, risk is carried by the wholesalers
- Delivery channels may need to be set up or modified.

### **10.4.12 Review Question 9**

For returning customers, improvements might include: a hyperlink in the introduction that leads them straight to the pages they have often visited in the past; an option to keep a record of their details so that they do not need to be re-entered on each visit; free offers for customers who make several purchases.

For marketing, multiple products improvements might include: a search engine for your products; easier access to products and demonstrations; use of a shopping trolley with a running cost total.

### **10.4.13 Review Question 10**

The four transformations of DAGMAR could be interpreted in the context of Internet commerce as follows:

1. Unawareness to awareness may be achieved by, among other things: judicious use of the META and TITLE HTML tags so that search engines find the page; using synonyms carefully to maximise hit rates; adding a link to the page from those pages popular with the target audience.
2. Awareness to comprehension may be achieved by providing suitable information about your service or product, including descriptions, statistics and pictures, as appropriate.
3. Comprehension to conviction may be achieved by offering samples of the product or service. If the customer provides information through, for example, a questionnaire, this may be a sign of increased confidence.
4. Conviction to action is sometimes perceived to have happened when the customer receives samples of the product. However, from the producer's viewpoint, this transformation only occurs when the customer actually purchases an item.

# Chapter 11. Basic Issues in Web Security

## Table of Contents

Objectives 1	
11.1	Introduction to Web Security..... 1
11.1.1	Why the Internet is Insecure? ..... 1
11.1.2	Why make information secure? ..... 2
11.2	Common vulnerabilities ..... 2
11.2.1	SQL Injection..... 2
11.2.2	Buffer Overflow..... 3
11.2.3	Sensitive Data Exposure ..... 3
11.2.4	Broken Authentication and Session Management..... 3
11.2.5	Security Misconfiguration..... 3
11.3	Web Security Solutions ..... 3
11.3.1	HTTPS 3
11.3.2	Certificates ..... 4
11.3.3	Encryption..... 4
11.4	Discussion ..... 4

## Objectives

At the end of this chapter you will be able to:

- Understand the need for web security;
- Understand some of the common web security vulnerabilities; and
- Understand some of the web security solutions.

## 11.1 Introduction to Web Security

When information transmitted over the web, not only does the data have reach its destination, but it needs to arrive intact and uncorrupted (**integrity**), and other people should be prevented from seeing it (**confidentiality**). The nature of the Internet makes directing information to reach its destination relatively trivial, but ensuring its integrity and confidentiality is more difficult. Fortunately, encryption algorithms have made both integrity and confidentiality feasible. Additionally, users like to know that the information they receive is genuine (**authentication**) and that the sender of the information cannot deny that they sent it (**non-repudiation**).

The web is an interconnection of networks. Everybody uses the Internet to transfer data and that the data has value (and cost), and so it is a subject to theft. Types of information that are stolen include personal user's information, commercial or technical data (including commercial secrets and intellectual property), or even security and military information. Leaking of such information can stay undiscovered for months, if not years, doing damage to people that sent information and also to third parties.

### 11.1.1 Why the Internet is Insecure?

One of the main reasons for such vulnerabilities is the fact that web application developers are often not very well versed with secure programming techniques. As a result, security of the application is not necessarily one of the design goals. This is exacerbated by the rush to meet deadlines in the fast-moving e-commerce world.

The Internet is a packet-passing network, and so information sent from one machine to another passes through

## Web Security

many intermediate machines as the data is routed towards its destination. These intermediate machines can see all the packets routed through them, as well as keep copies of the packets and possibly change their data content before passing them on. Information on a network or internetwork is clearly not confidential by nature.

It also means that the information's receiver cannot be sure that the information has been unchanged: in other words, there are doubts about the information's integrity.

As any intermediate machine may have changed the data, the data can also not be authenticated, and the original source can deny that they originally sent the data (they can repudiate the data).

While some of these problems are alleviated due to the nature of the Internet (since the various packets containing the data may go via different routes), they cannot be eliminated.

### 11.1.2 Why make information secure?

While a large portion of information on the Internet is meant to be widely shared (such as a company's website), there is also important information transmitted over the Internet that is meant to be private and secure.

Consider the needs of e-commerce, where private information, such as credit card details, are transmitted online.

When consumers purchase goods via credit card, they do not want any intermediate people to know their credit details.

Generally, any important information sent over the Internet should be secured in some way. There are obviously different types of information, and some need more security than others.

The important issues around obtaining a credit card number from a customer are:

- If the transmission of the credit card details isn't confidential, customers are open to credit card fraud.
- If the data's integrity isn't assured, then their credit details, or their purchase information, may be invalid.
- If the communication details cannot be authenticated, then there is no guarantee that the purchased products are being sent to the right person.
- If there is no non-repudiation, the customer can deny that they ordered the product once they have received it, and cancel the credit card payment.

#### Exercise 1

How would you go about trying to make messages secure? Consider the realm of popular books and movies, can you think of any examples where information is made secure? What were the means used to secure information in your examples?

You can find a discussion of this exercise at the end of the chapter.

## 11.2 Common vulnerabilities

Some of the top five security vulnerabilities are SQL injection; buffer overflow; sensitive data exposure broken authentication and session management; and security misconfiguration<sup>1</sup>. These are briefly discussed below.

### 11.2.1 SQL Injection

SQL injection is a technique where malicious users can inject SQL commands into an SQL statement, via web page input. Injected SQL commands can alter SQL statement and compromise the security of a web application.

A Database is the heart of many web-applications and is used to store information needed by the application, such as, credit card information, customer demographics, customer orders, client preferences, etc. SQL Injections happen when a developer accepts user input that is directly placed into a SQL Statement and doesn't properly validate and filter out dangerous characters. This can allow an attacker to alter SQL statements passed to the

---

<sup>1</sup> <http://resources.infosecinstitute.com/the-top-five-cyber-security-vulnerabilities-in-terms-of-potential-for-catastrophic-damage/>

database as parameters and enable her to not only steal data from your database, but also modify and delete it.

A database is vulnerable to SQL injections when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed. SQL injection attacks are also known as SQL insertion attacks.

### 11.2.2 Buffer Overflow

A buffer overflow vulnerability condition exists when an application attempts to put more data in a buffer than it can hold. Writing outside the space assigned to buffer allows an attacker to overwrite the content of adjacent memory blocks causing data corruption, crash the program, or the execution of an arbitrary malicious code

### 11.2.3 Sensitive Data Exposure

Sensitive data exposure occurs every time a threat actor gains access to the user sensitive data. Data could be stored (at rest) in the system or transmitted between two entities (i.e. servers, web browsers), in every case a sensitive data exposure flaw occurs when sensitive data lack of sufficient protection. Sensitive data exposure refers the access to data at rest, in transit, included in backups and user browsing data. The attacker has several options such as the hack of data storage, for example by using a malware-based attack, intercept data between a server and the browser with a Man-In-The-Middle attack, or by tricking a web application to do several things like changing the content of a cart in an e-commerce application, or elevating privileges.

### 11.2.4 Broken Authentication and Session Management

The exploitation of a broken Authentication and Session Management flaw occurs when an attacker uses leaks or flaws in the authentication or session management procedures (e.g. Exposed accounts, passwords, session IDs) to impersonate other users. This kind of attack is very common.

### 11.2.5 Security Misconfiguration

Below some typical example of security misconfiguration flaws:

- Running outdated software.
- Applications and products running in production in debug mode or that still include debugging modules.
- Running unnecessary services on the system.
- Not configuring problems the access to the server resources and services that can result in the disclosure of sensitive information or that can allow an attacker to compromise it.
- Not changing factory settings (i.e. default keys and passwords).
- Incorrect exception management that could disclose system information to the attackers, including stack traces.
- Use of default accounts.

## 11.3 Web Security Solutions

Two main tasks of any web security solution is to:

- Provide correct identification of the remote side in network conversation; and
- Prevent third parties that have possibility to access the network, over which the data is transmitted, from accessing the data being sent.

Some of the web security solutions that allow the client and the server to securely exchange information and minimize the chance for attack are described below.

### 11.3.1 HTTPS

There are several widely used protocol schemes available. They are SSH (Secure Shell) and SSL (Secure Socket Layer/Transport Level Security). Both protocols work on transport network level ("above" TCP protocol) and utilize similar schemes. SSL is more widely used because of its adoption for secure WWW data transfer. Both protocols provide transparent security; this allows use of standard Internet protocols over SSL or SSH.

The most well-known application for SSL protocol is securing commercial Internet communications. Most of commercial web sites offer an option (or even force) for use of SSL, which is used for HTTPS protocol. This is however not the only protocol to use SSL. Actually most TCP-based protocols (like POP3 and IMAP for mail, NNTP for news etc.) can work over SSL. SSH is also used to provide security for FTP and shell protocols.

### HTTPS

Hypertext Transfer Protocol Secure or Hypertext Transfer Protocol over SSL is used for secure communication over a network, or perhaps more importantly – over the Internet. You would see https:// in the URL and a lock icon in the browser when you access a page that uses HTTPS. Hyper Text Transfer Protocol Secure (HTTPS) is the secure version of HTTP, the protocol over which data is sent between your browser and the website that you are connected to. HTTPS is often used to protect highly confidential online transactions like online banking and online shopping order forms.

Consider developing an e-commerce website that requires your users to enter sensitive information, such as credit card details, in order to proceed with an online transaction. If the information travels over the Internet as is and is intercepted by someone, it could be easily understood and misused. This is where HTTPS comes in – if you need to prevent these types of threats, you need to go HTTPS.

HTTPS promises you two things; first, the sensitive data is encrypted into gibberish by applying a cryptography mechanism which can be decrypted only by your server, the certificate owner. Now, if this information is intercepted with a man-in-the-middle attack, it will be meaningless. Secondly, HTTPS authenticates that the website is the website it claims to be. In your case, it will validate your website before sending your user's encrypted credit card details so no one else can imitate you.

Thus, going HTTPS authenticates your website and protects sensitive information being communicated over the Internet. This is made possible with the help of Certificates and Encryption.

### 11.3.2 Certificates

In order for you to go HTTPS, you need a Certificate. It is a digital document that your website submits to proclaim your identity to the user, the web browser. The certificates are issued by companies known as Certificate Authorities (CA) which will encrypt your web related information such as your domain name, server platform and identity information such as company's name address, phone number etc. within the certificate. You may wonder how a browser would trust a certificate. All browsers come with a set of pre-installed information letting them know of trusted certificate authorities. When you go HTTPS, you'll have your certificate in your server which will be sent to your user whose browser will certify you.

### 11.3.3 Encryption

We know that HTTPS encrypts data before sending it over the Internet and the server decrypts it. In the encryption-decryption scenario, a pair of keys is involved. One is public and the other is private. When your website wants your user to send information, your server instructs the user's browser with a key (public) to encrypt the data which is to be sent over. Once the encrypted message is received, the server will use its private key to decrypt and understand the data. In HTTPS, any plain text encrypted with the public key can only be decrypted by the holder of the private key.

## 11.4 Discussion

### EXERCISE 1

Both these examples are based upon securing verbal information by means of a code.

1. In the movie Mission Impossible, the hero breaks into the CIA's headquarters and steals a list of all the undercover agents around the world. This is obviously very important information and it should be kept secure. The security measures the CIA used were extensive, and required a lot of effort to overcome them.
2. In the book, Enigma, by Robert Harris — a novel based closely on Turing and the work at Bletchley Park in the UK during the Second World War. The 'Enigma' machine was a mechanical coding device used to pass messages between U-boats and their command headquarters. In times of war, secure communication is perhaps extra important. The cracking of the code at Bletchley Park was greatly assisted by the capture of an Enigma machine and various decoding pamphlets from an abandoned German U-boat. Alan Turing, the mathematician and computer pioneer, was a leading figure in this work.

# Chapter 12. JavaScript 1: Basic Scripting

## Table of Contents

Objectives.....	2
11.1 Introduction.....	2
11.1.1 Differences between JavaScript and Java.....	2
11.2 JavaScript within HTML.....	3
11.2.1 Arguments.....	5
11.2.2 Accessing and Changing Property Values.....	5
11.2.3 Variables.....	6
11.2.4 JavaScript Comments.....	8
11.3 Some Basic JavaScript Objects.....	10
11.3.1 Window Objects.....	10
11.3.2 Document Object.....	12
11.3.3 Date Objects.....	13
11.4 Review Questions.....	17
11.4.1 Review Question 1.....	17
11.4.2 Review Question 2.....	18
11.4.3 Review Question 3.....	18
11.4.4 Review Question 4.....	18
11.4.5 Review Question 5.....	18
11.4.6 Review Question 6.....	18
11.4.7 Review Question 7.....	18
11.4.8 Review Question 8.....	18
11.4.9 Review Question 9.....	18
11.4.10 Review Question 10.....	19
11.4.11 Review Question 11.....	19
11.5 Discussions and Answers.....	19
11.5.1 Discussion of Exercise 1.....	19
11.5.2 Discussion of Exercise 2.....	19
11.5.3 Discussion of Exercise 3.....	20
11.5.4 Discussion of Exercise 4.....	20
11.5.5 Activity 2: Checking and Setting Background Colour.....	20
11.5.6 Activity 3: Setting a document's foreground colour.....	21
11.5.7 Activity 4: Using user input to set colours.....	21
11.5.8 Activity 5: Dealing with errors.....	22
11.5.9 Activity 6: The confirm method.....	23
11.5.10 Activity 7: Changing the window status.....	24
11.5.11 Activity 8: Semicolons to end statements.....	24
11.5.12 Activity 9: including separate JavaScript files.....	24
11.5.13 Activity 10: Opening a new Window.....	24
11.5.14 Answer to Review Question 1.....	25
11.5.15 Answer to Review Question 2.....	25
11.5.16 Answer to Review Question 3.....	25
11.5.17 Answer to Review Question 4.....	25
11.5.18 Answer to Review Question 5.....	25
11.5.19 Answer to Review Question 6.....	25
11.5.20 Answer to Review Question 7.....	25
11.5.21 Answer to Review Question 8.....	25
11.5.22 Answer to Review Question 9.....	26
11.5.23 Answer to Review Question 10.....	26
11.5.24 Answer to Review Question 11.....	26

# Objectives

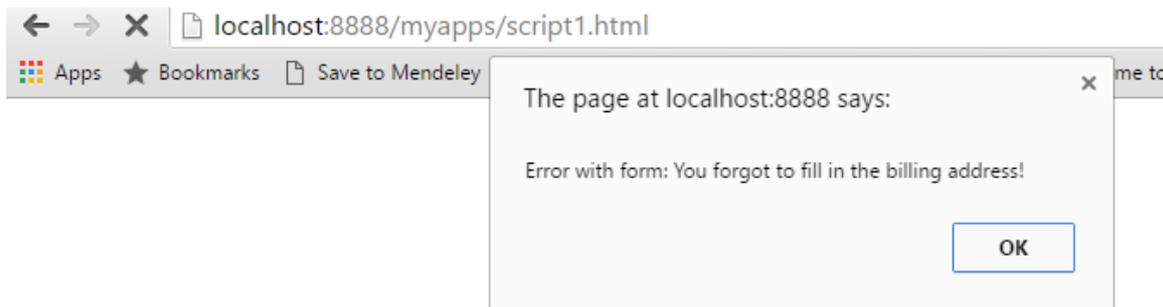
At the end of this chapter you will be able to:

- Explain the differences between JavaScript and Java;
- Write HTML files using some basic JavaScript tags and objects.

## 11.1 Introduction

Web browsers were originally designed to interpret HTML with two primary purposes: to render documents marked up in HTML to an acceptable level of quality, and, crucially, to be able to follow hyperlinks to resources. As the Web grew, so did the demand for more sophisticated Web content. Among many other extensions, graphics, forms, and tables were added to the HTML standard. With the exception of forms, there is nothing in HTML that supports interaction with the user. Given the ubiquity of Web browsers, and the effort which millions of ordinary people have put into learning to use them, they provide an almost universal starting point for interacting with complex systems, particularly commercial, Internet based systems. Hence the need for sophisticated interaction facilities within Web browsers.

The main means for providing interactivity within HTML documents is the JavaScript programming language. HTML documents can include JavaScript programmes that are interpreted (i.e. run) by the Web browser displaying the Web document. In a real sense, JavaScript allows a Web document to interact with its environment — that is, with the browser that is displaying it. Ultimately, it lets the Web document become more interactive, to the user's benefit. For example, the following message could be given to a user when they submit a form with a missing field:



The above message can be shown with the following JavaScript code.

```
<SCRIPT>
window.alert('Error with form: You forgot to fill in the billing address!')
</SCRIPT>
```

The JavaScript code is contained within the `<SCRIPT>` and `</SCRIPT>` tags. Everything between those tags must conform to the JavaScript standard (the standard itself is an ECMA International standard, called ECMAScript). The above statement is an instruction to the browser requesting that an alert box display the message "Error with form: You forgot to fill in the billing address!".

This unit will later cover another way to include JavaScript in HTML documents. It is worth noting for now that the `<SCRIPT>` tag can include a language attribute to ensure the browser interprets the enclosed commands as JavaScript, since other languages have, in the past, been used (such as VBScript, which is no longer used in new websites, and is supported by very few browsers). For simplicity, we will use the attribute's default value (of JavaScript) by omitting the attribute from the `<SCRIPT>` tag.

### 11.1.1 Differences between JavaScript and Java

While programming is covered in the programming module of this course, JavaScript differs from Java in some important areas that we will quickly review. JavaScript objects are covered in more detail in later chapters, so we will not go into any great depth here.

In Java, all functions must belong to a class, and for this reason are called methods. In JavaScript, a function does

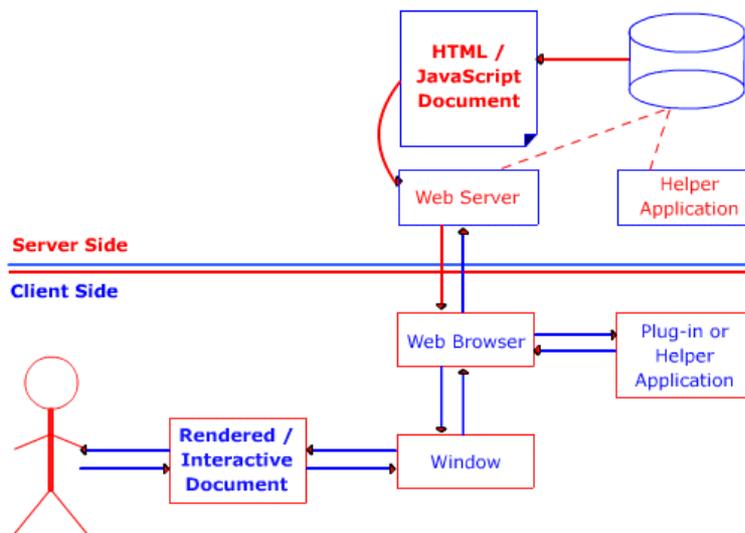
## JavaScript 1: Basic Scripting

not have to belong to a particular object at all. When a function does, however, it is often called a method. Functions and methods are both implemented in the same way, using the function keyword. All methods are functions, but not all functions are methods.

Unlike Java, JavaScript does not contain the idea of classes. Instead, JavaScript has constructors, which are a special kind of function that directly creates objects. These constructor functions define the state variables which each object holds and initialises their values. These variables are often called the object's properties. Constructor functions also supply objects with their methods.

In JavaScript, functions are themselves a special kind of object called Function Objects. Function Objects can be called just as normal functions in other languages, but because they are objects they can themselves be stored in variables and can be easily passed around as, say, arguments to other functions. They can also contain properties of their own. Constructor functions have a special Prototype property which is used to implement inheritance, as will be explained later in the chapter on objects. Constructor functions are called using the new keyword when creating objects.

JavaScript communicates with its environment by calling methods on objects representing components of that environment, such as an object representing the window the HTML document is displayed in, an object representing the document itself, and so on. Ignoring the server side at present, we conceptually have a browser that interacts with a window, which interacts with a document, which is itself the user interface. JavaScript allows the user interface to be programmed. This is accomplished by providing the means, in JavaScript, for a user to interact with a system via the document rendered in the browser window, as depicted below.



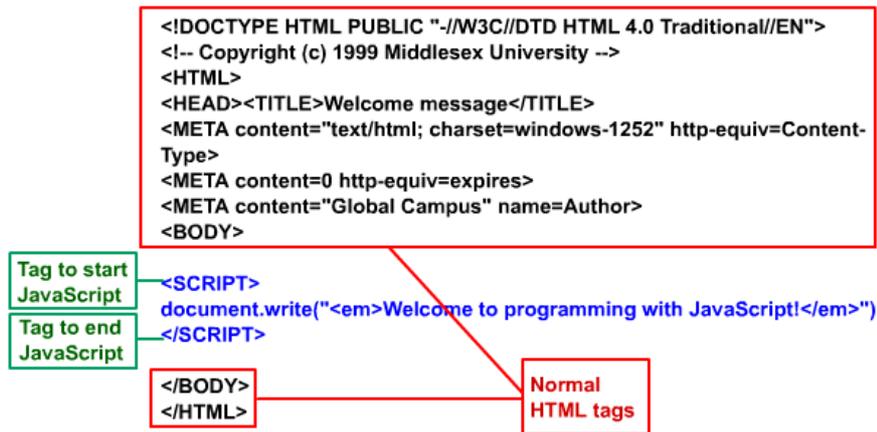
A user may request a document via an URL; a Web server delivers the document to a Web browser which not only displays it, but also executes any interactive elements.

Now do Review Questions 1, 2 and 3.

## 11.2 JavaScript within HTML

JavaScript statements are embedded within an HTML document and are interpreted by a Web browser. Unlike programming in Java, a programmer does not programme with JavaScript by preparing source code and compiling it to produce executable code. JavaScript is directly executed by the Web browser. The most general form of JavaScript used in HTML documents is to include JavaScript statements within `<SCRIPT>` and `</SCRIPT>` tags. Each JavaScript statement is written on a separate line. If a statement is in some way incorrect, the browser (and its built-in JavaScript interpreter) may report an error, or it may stop executing the erroneous JavaScript and do nothing. Let us examine a simple HTML4.0 document that does nothing that could not be done with HTML alone. All that it does is have the document include the italicised text "Welcome to programming with JavaScript!".

## JavaScript 1: Basic Scripting



Most of the document is normal HTML. There are two new tags — `<SCRIPT>` to indicate the start of JavaScript code, and `</SCRIPT>` to indicate its end. The only line of JavaScript is the one containing:

```
document.write(" <em>Welcome to programming with
JavaScript!</em> ")
```

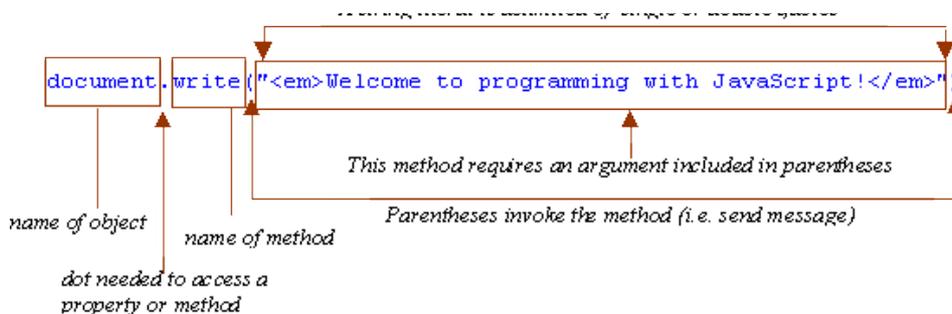
### Activity 1

1. Using HTML5 tags create a file named `script1.html` with `<HTML>` and `<BODY>` tags.
2. Insert the following code and load the file to your browser.

```
<script>
document.write(" <em>Welcome to programming with
JavaScript!</em> ")
</script>
```

What this mixture of HTML and JavaScript does is to make the browser switch from interpreting HTML to interpreting JavaScript when it encounters the `</SCRIPT>` tag. The line is interpreted as follows: first, it calls the `write` method of the `document` object with the argument `"<em>Welcome to programming with JavaScript!</em>"`. The argument is a literal string containing valid HTML markup between the start and end emphasis tags (`<em>` and `</em>`) that cause what they delimit to appear in italics.

Let us examine the individual parts of the expression, as in the diagram below. Note that in this case the argument is a literal string.



As you might guess, all JavaScript does in this statement is to insert the argument text 'into' the document to be interpreted as HTML — hence the emphasis tags produce italics. Of course, the original document delivered by the server is not modified; nothing is actually written into the original document. What happens is that the JavaScript method inserts its argument in the stream of characters that the browser is interpreting.

Now carry out the following exercise and once you have studied its discussion continue with the unit's content.

### Exercise 1

Modify the document in Activity 1 so that some HTML text is outputted immediately before and immediately after

the welcome greeting.

You can find a discussion of this exercise at the end of the unit.

## 11.2.1 Arguments

Arguments are also known as parameters. An argument is information that must be included with a function or method so that it may achieve its purpose. As in Java, some methods and functions are designed to have information supplied to them, so that their effect can differ in detail. If a function or method is designed in this way, it must be supplied with the required arguments when it is called. The document method `write` has been designed and implemented that way. Another function that needs a string argument is the window method `alert`, which was briefly shown earlier in these notes. The next exercise makes use of `alert`.

### Exercise 2

Write the JavaScript statements that will show the welcome greeting in an alert box. (Hint: remember `alert` is a method of `window` objects, not of `document` objects.)

You can find a discussion of this exercise at the end of the unit.

`document.write` is an example of a function that can take an unspecified number of arguments. It must, however, have at least one. The following example uses more than one argument:

```
document.write("<BU>", "<LI>books</LI>", "<LI>magazines</LI>", "</BU>")
```

In JavaScript, as in Java, multiple arguments to a function must be separated by commas and provided in an order determined by the function. There is no simple rule to tell how many arguments a method must have or what their order should be, and this information can only be obtained from the method's definition, or from a good reference document or book.

### To Do

Find a JavaScript Reference source that has information about the objects, properties and functions available to a JavaScript programmer.

Now do Review Questions 4, 5 and 6.

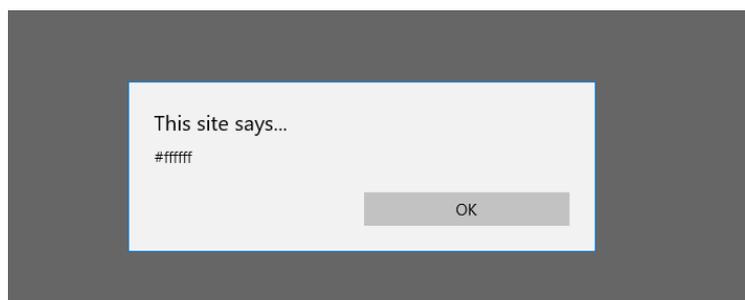
## 11.2.2 Accessing and Changing Property Values

In JavaScript, it is commonplace to directly access or change the value of a property. One such document property is `bgColor`, which represents the document's background colour. Directly accessing a property is similar to accessing a method: write the object name followed by a period (a dot) and the property name. Thus, to access a document's background colour, write `document.bgColor`. Note the American English spelling of 'Color', and the use of a capital 'C' inside the word. The spelling and capitalisation must be precise or a browser's JavaScript interpreter will not properly process the script.

Of course, methods can be combined with the ability to access properties. For example, the `alert` method of `window` can be used to report the document's background colour:

```
window.alert(document.bgColor)
```

This would produce a dialogue box as follows.



The box shows the hexadecimal (HEX) representation of the colour, with `#FFFFFF` being the HEX for white, a document's default background colour.

The content of the dialogue box can be made more comprehensible by announcing the value with another string explaining the value. One way to do this is with the string concatenation operator, `+`, which combines what precedes it with what follows it. So, for example, `"changing " + "colour"` results in the string `"changing colour"`. Note the space at the end of the first string literal appears between the two words in the string resulting from using `+`. Hence you can concatenate `"background colour is (in Hex): "` with `document.bgColor` to produce more helpful output:

```
window.alert("background colour is (in Hex): " + document.bgColor)
```

To change a document's background colour, simply assign the new value to the property using the `=` operator. Do not confuse this symbol with equality. JavaScript belongs to a class of programming languages (just as Java) that use `=` as the assignment operator: it assigns the value of whatever is to its right to whatever is on its left. Another symbol is used for equality testing, as we discuss later. Hence, to set the background colour of a document to blue (HEX 0000FF), we can use:

```
document.bgColor = "0000FF"
```

JavaScript provides a set of mnemonic strings for many colours. For example:

```
document.bgColor = "blue"
```

Note that no matter what value was used to set a colour, if you access it as above the output will always be in HEX. (This is because JavaScript automatically converts between different types of data. We will see this a lot in JavaScript programming.)

### Activity 2: Checking and Setting Background Colour

Write and test a document containing HTML and JavaScript that displays the background colour of the document, warns you of a change of colour is to take place, then changes it to blue (use the string `"blue"`). Repeat this for the orange-like colour coral, using its HEX representation `"FF7F50"` and confirm the change with an alert.

You can find a discussion of this activity at the end of the unit.

### Activity 3: Setting a Document's foreground colour

JavaScript considers the default colour of a document (i.e. the colour of its text) to be the property `fgColor`. Write a script to set the background colour of a document to blue and the foreground colour to white.

You can find a discussion of this activity at the end of the unit.

## 11.2.3 Variables

With the exception of alert dialogue boxes, so far we have used JavaScript to do nothing more than what HTML can do. We now proceed to the vital concept of programming with variables via another interactive facility — dialogue boxes that allow the user to enter data.

### Exercise 3

An important principle of object-oriented programming is that any facilities provided by an object should be encapsulated within the object. Remembering which object provides the

## JavaScript 1: Basic Scripting

*alert* method, which object do you think has a method for prompting the user for input in a dialogue box.

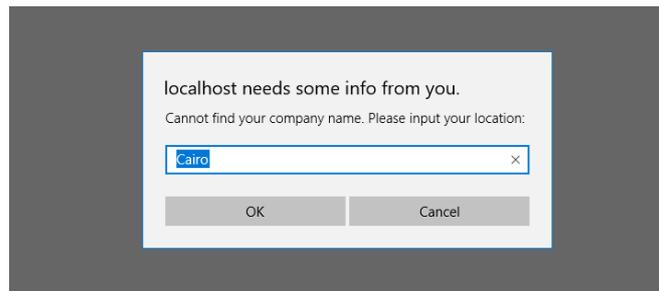
You can find a discussion of this exercise at the end of the unit.

The *prompt* method allows you to make a window display a dialogue box containing an explanation prompt and a default value for input.

For example, the statement below explains to the user that their company name cannot be found (let's assume some previous processing has determined this). It provides a suggested default of Cairo, which we assume makes sense for the Web application.

```
window.prompt("Cannot find your company name. Please input your location:", "Cairo")
```

Note that the two string arguments, each delimited by their own quotes, are separated using a comma. The output from a document containing this JavaScript appears as run on Microsoft Edge.



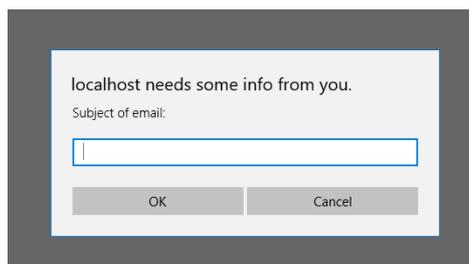
So, what might be done with input obtained this way? In general, we would use it to change the state of some part of the Web application, and such a simple change of state could eventually lead to more complex behaviour. One straightforward use is to modify the HTML stream to include the input obtained by *window.prompt*. For example, imagine an application that generated an email message whose content is an HTML document. We could ask for the subject and then write it as a level 1 heading (i.e. using `<H1>` and `</H1>`):

```
document.write("<H1>Subject: "+window.prompt("Subject of email:", "")+"</H1>")
```

This rather complicated argument to `document.write` needs explaining. The complication comes from needing to evaluate the prompt to determine the argument to write as a whole. When this statement is executed, the interpreter tries to concatenate the three strings:

1. `"<H1>Subject: "`
2. `window.prompt("Subject of email:", "")`
3. `"</H1>"`

String 1 is the first part of the heading; it includes the start tag for the level 1 heading, and the beginning of the heading — 'Subject: '. String 3 is the end tag for the level 1 heading. String 2 cannot be part of the concatenation for the *write* method unless it is itself evaluated; hence the following dialogue box is produced. Notice how no default for the input is provided, because the second argument is an empty string — a string with zero characters in it.



If the user types in 'Unable to book flight' and clicks OK (or presses the Enter key), then the *prompt* method delivers that string object as String 2 and the concatenation can be completed, resulting in the heading in the browser below:



An alternative way to programme this type of behaviour is to break it up into smaller pieces and to provide the means to remember the result of evaluating some complex expression, then using that result later in the programme in whatever way is needed. This is what variables are for: they are names that can be used to refer to an object or value for a variety of purposes. They are called variables because they can be used to refer to different objects at different times of the programme's execution. As it happens, in JavaScript a variable can refer to objects of any type, whether they be strings, numbers, or user defined objects. This is very flexible and powerful, but it means that the interpreter cannot help you by restricting the functionality of the variable to a declared purpose, as in some other languages (such as Java). The most general way to use a variable is to declare it with a special key word, *var*, as in:

```
var input
```

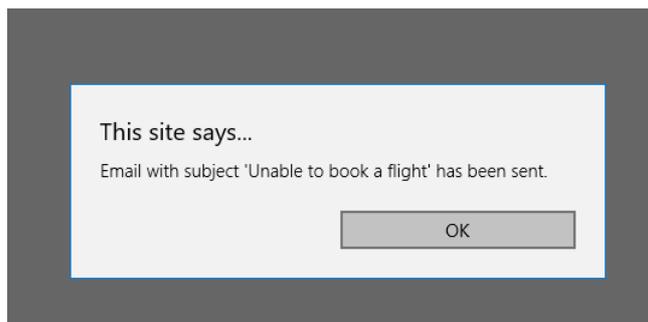
Once a variable has been declared it can be made to refer to an object with the assignment operator, =, introduced earlier. Now the email subject prompter can be reprogrammed as follows (below, we choose the obvious variable name — *input* — which has nothing to do with the HTML tag or the JavaScript object):

```
<SCRIPT>
var input
input = window.prompt("Subject of email:", "")
document.write("<H1>Subject: " + input + "</H1>")
</SCRIPT>
```

Executing this script has the same effect as the earlier version that included the prompt as an argument to *write*. A user cannot tell the difference.

#### Exercise 4

Write another script to be included in the same document as the previous one so that the new script produces a confirmation that the email with the given subject has been sent, as in the following dialogue box.



You can find a discussion of this exercise at the end of the unit.

## 11.2.4 JavaScript Comments

In the same way that HTML provides the means to prevent an HTML interpreter (a browser) from acting on text in a document, JavaScript also provides the means to turn its statements into comments that the JavaScript interpreter ignores. Because of its C++, C and Java heritage, JavaScript accepts comment syntax from all of these languages: anything between */\** and *\*/* is ignored, as is everything on a line following *//*. The former brackets a comment, which can be over several lines or within a statement. The latter is typically used when the end of a line contains a comment. For example, the above script can be commented as follows:

```
<SCRIPT>
var input // this variable will be used in a later script
input = window.prompt("Subject of email:", "")
```

## JavaScript 1: Basic Scripting

```
/* We have to generate HTML according to company style
guidelines found in
Document 1998-EM-GD-V3.1, hence what follows. */
document.write("<H1>Subject: " + input + "</H1>")
</SCRIPT>
```

JavaScript is used within HTML and it is possible that older browsers are unable to run JavaScript, or that a user has disabled the JavaScript functionality of a recent browser. Therefore JavaScript also understands the opening HTML comment, but not the closing comment. This allows JavaScript code to use a mixture of HTML comments to allow the JavaScript to be ignored by browsers that cannot run JavaScript and would format it as HTML text.

The previous example could be changed so that it works in a JavaScript-enabled Web browser but has no effect on the output of a Web browser that cannot handle JavaScript. The line immediately after the <SCRIPT> tag is the start of an HTML comment, which is ignored by both language interpreters. The line before the </SCRIPT> tag is ignored by a JavaScript interpreter, but if there was none, the HTML interpreter would have been ignoring everything up to the closing HTML comment bracket.

```
<SCRIPT>
<!--the JavaScript can be ignored
var input // this variable will be used in a later script input =
window.prompt("Subject of email:", "")
/* We have to generate HTML according to company style guidelines
found in
Document 1998-EM-GD-V3.1, hence what follows. */
document.write("<H1>Subject: " + input + "</H1>")
//--->
</SCRIPT>
```

### Activity 4: using user input to set colours

Write and test an HTML/JavaScript document to prompt the user for background colour and foreground colour and then output sample text. You should use appropriately named variables and maybe reuse them when generating the sample text. When testing, make sure you have a variety of inputs — valid and invalid colours, and cancelled inputs. Note down what happens in each case.

You can find a discussion of this activity at the end of the unit.

### Activity 5: dealing with errors

The following is bad JavaScript. Type (or copy) it into a suitable HTML document to explore what goes wrong when you try to interpret this in a browser. Correct each problem statement as you see necessary and write down what you did and why. (Hint: look at the use of the *write* method from which you can infer what the script is to do.) Also identify anything that seems unnecessary and anything you think is wrong at first sight but which might turn out to be correct. Note that different browsers (different JavaScript interpreters) will report different problems. If you have access to more than one browser you may like to see which provides most help.

Note that you are not expected to understand all the problems in the script below. What you are expected to understand is that you can 'instrument' the code with suitable *write* methods or *alert* methods (like the ones you have already used) to help you reason about what is happening (or not happening). So be prepared to temporarily add statements to the code so you can trace how it is being interpreted. And be prepared to spend quite a bit of time exploring the code.

```
var fore, back
back = document.bgcolor
fore = document.fgColor = fore
newFore = window.prompt("Foreground colour:", fore)
document.fgColor = newFore
input = window.prompt("What do you want written out
underlined in colour document.write("You typed: " + <U> +
"input + </U>")
```

You can find a discussion of this activity at the end of the unit.

## Activity 6: The confirm method

As well as the *alert* and *prompt* methods, the *window* object provides a *confirm* method that produces a dialogue box with the buttons OK and Cancel. Like *alert*, it takes a string argument that is displayed in the dialogue box. Explore this method by writing HTML/JavaScript documents that use it.

You can find a discussion of this activity at the end of the unit.

Now do Review Questions 7, 8 and 9.

# 11.3 Some Basic JavaScript Objects

So far we have used the *Window* and *Document* objects. We now list many of the most frequently needed properties of *Window* and *Document* and all of the details of *Date*, which we have not yet used.

You do not have to memorise these lists. You will eventually remember what you most often use. However, it is important to spend time reading reference material because sometimes you will need some facility and will need to be aware if it exists or not. Later activities assume you can use most of the facilities described below.

Notice that some properties are described as an 'array' with square brackets. An array is a collection of items numbered from zero, as in Java, and is accessed in the same way, using a suffix containing the index number in square brackets. For instance, *window.frames[2]* refers to the third frame in a window. Arrays are covered in more detail in a later chapter.

First, note that variables *window* and *document* are automatically declared so that we have been able to make use of them when scripting. They refer to the current window and its document respectively. Automatic declaration, however, is not usual. For example, to use a *Date* object you will have to create one with a constructor, as described later.

Note that where a method requires an argument or arguments, these arguments are indicated by a phrase in angle brackets that list the arguments that are needed. For example, *<string>* would indicate that a string argument is needed. (Do not confuse these with HTML tags.)

## 11.3.1 Window Objects

The following table lists frequently used properties and methods that are supported by Firefox, Internet Explorer and Opera.

Window	Description
Properties (Partial List)	<p><b>defaultStatus</b> — a string that specifies what appears in the window's status area</p> <p><b>document</b> — a reference to the document object contained by the window</p> <p><b>frames[]</b> — an array of frames in the window's document</p> <p><b>length</b> — the number of elements in the frames array, i.e. the number of frames</p> <p><b>location</b> — a reference to the Location object for the window</p> <p><b>Math</b> — a reference to a mathematical object that provides various mathematical functions</p> <p><b>name</b> — a string containing the name of the window</p>

Window	Description
	<p><b>self</b> — a reference the window itself</p> <p><b>status</b> — a reference to the window's status area</p> <p><b>top</b> — a reference to the top-level window containing this one, only if this one is a frame</p>
Methods (Partial List)	<p><b>alert(&lt;string&gt;)</b> — produced a dialogue box containing the string and a single button labelled OK</p> <p><b>close()</b> — close the window</p> <p><b>confirm(&lt;string&gt;)</b> — ask a yes/no question with the string argument</p> <p><b>moveBy(&lt;number in x&gt;, &lt;number in y&gt;)</b> — move the window by the given number of pixels in the x (horizontal) and y (vertical) directions</p> <p><b>moveTo(&lt;number for x&gt;, &lt;number for y&gt;)</b> — move the window to the location given for x (horizontal) and y (vertical) directions</p> <p><b>prompt(&lt;string for prompt&gt;, &lt;string for default&gt;)</b> — prompt with OK and Cancel buttons using the first string as prompt and the second as default for input</p> <p><b>resizeBy(&lt;number in x&gt;, &lt;number in y&gt;)</b> — resize the window by the given number of pixels in the x (horizontal) and y (vertical) directions</p> <p><b>resizeTo(&lt;number for x&gt;, &lt;number for y&gt;)</b> — resize the window to the size given for x (horizontal) and y (vertical) directions</p> <p><b>scrollTo(&lt;number for x&gt;, &lt;number for y&gt;)</b> — scroll the document in the window so that the position given for the x (horizontal) and y (vertical) direction is in the tope left corner of the window</p>

Note that because the *Window* object (referred to by the *window* variable) is the main object of the system as far as JavaScript is concerned, its name can be left out. Therefore we could have simply written `confirm("Click on OK or Cancel or close box")`. However, we recommend you explicitly use the *window* object.

There are some exceptions to this recommendation. The first is the use of the *Math* object ('math' being the American abbreviation for 'mathematics'). Because the window contains a *Math* object as a property, you can refer to it simply using *Math*. For example `Math.floor(<number>)` allows you to truncate (i.e. round down) a real number to the next smaller integer (not the closest to zero). Hence:

```
document.write("Math.floor(6.7) = ", Math.floor(6.7), "<BR>")
document.write("Math.floor(-6.7) = ", Math.floor(-6.7), "<BR>")
```

produces the following:



## 11.3.2 Document Object

The list of *Document* properties given below is almost complete (excluding those specific to particular browsers). Many are arrays, or are associated with other objects not yet encountered. Do not expect to use these in this unit.

Document	Description
Properties (Partial List)	<p><b>aLinkColor</b> — string specifying colour of active links</p> <p><b>anchors[]</b> — an array of Anchor objects</p> <p><b>applets[]</b> — an array of Java (not JavaScript) applets (one for each HTML &lt;APPLET&gt; tag)</p> <p><b>bgColor</b> — background colour of document</p> <p><b>cookie</b> — a string associated with document cookies</p> <p><b>domain</b> — specifies the Internet domain that was the source of the document</p> <p><b>embeds[]</b> — an array of embedded objects (one for each HTML &lt;EMBED&gt; tag)</p> <p><b>fgColor</b> — foreground colour of document text</p> <p><b>forms[]</b> — an array of Form objects (one for each HTML &lt;FORM&gt; tag)</p> <p><b>images[]</b> — an array of Image objects (one for each HTML &lt;IMG&gt; tag)</p> <p><b>lastModified</b> — a string specifying the date of the last change to the document as reported by its Web server</p> <p><b>linkColor</b> — string specifying colour of unvisited links</p>

Document	Description
	<p><b>links[]</b> — an array of Link objects (one for each hypertext link)</p> <p><b>location</b> — a synonym for the URL property (not the same as window.location)</p> <p><b>plugins[]</b> — a synonym for the embeds[] property</p> <p><b>referrer</b> — a string specifying the URL of the document containing the link to this one that the user followed to reach the document</p> <p><b>title</b> — title of the document (may not be changed)</p> <p><b>URL</b> — a string containing the URL of the document (may not be changed)</p> <p><b>vlinkColor</b> — string specifying colour of visited links</p>
Method (Partial List)	<p><b>close()</b> — closes the document stream opened using the <i>open()</i> method</p> <p><b>open()</b> — open a stream to which document contents can be written, i.e. to which output of subsequent <i>write()</i> or <i>writeln()</i> methods will be appended</p> <p><b>write(&lt;at least one argument&gt;)</b> — appends to current output stream; takes any argument that can be converted to a string</p> <p><b>writeln(&lt;at least one argument&gt;)</b> — same as <i>write()</i> but adds a new line to output that usually has no effect for HTML, except if after &lt;PRE&gt; tag.</p>

### 11.3.3 Date Objects

*Date* objects have properties and methods for handling date and time. To access dates or times a *Date* object must be created, typically using *new Date()*. Facilities are provided for both the local date and time and universal date and time (UTC) or Greenwich Mean Time (GMT). For example, the following JavaScript code creates a new *Date* object as part of the initialization of the variable *today*.

```
var today = new Date()
document.write('1 -- ', today, '<BR>')
document.write('2 -- ', today.toGMTString(), '<BR>')
document.write('3 -- ', today.toLocaleString(), '<BR>')
```

The following is the output that results in Microsoft Edge.

## JavaScript 1: Basic Scripting

```

1 -- Tue Dec 08 2015 11:11:35 GMT+0200 (South Africa Standard Time)
2 -- Tue, 08 Dec 2015 09:11:35 GMT
3 -- 12/8/2015 11:11:35 AM

```

The following table lists the attributes and methods of *Date*. Note that *Date* has no properties. Usually you can interchange the terms attribute and property. In English they mean more or less the same thing. However, in object technology an attribute is simply an idea you name because you expect it is somehow represented by the object. Often an attribute is represented in JavaScript as a property, but not in this case. You cannot directly access attributes of a date like the hour or minute; you must use accessor methods, such as the methods that begin with 'get' or 'set' below.

Date	<i>Date</i> objects have attributes and methods for handling date and time. To access dates or times a <i>Date</i> object must be created, typically using <i>new Date()</i> . Facilities are provided for both the local date and time and universal date and time (UTC) or Greenwich Mean Time (GMT).
<i>Date</i> Constructors	<p>There are four forms of constructor for <i>Date</i></p> <ol style="list-style-type: none"> <li><b>new Date()</b> In the first form, a <i>Date</i> object is created initialised to the current date and time.</li> <li><b>new Date(time in milliseconds)</b> In the second form, the date and time given to the object created is specified by the time in milliseconds from 1 January 1970, GMT. This is usually used when the number of milliseconds has been computed using various <i>Date</i> methods.</li> <li><b>new Date(string in date format)</b> In the third form an appropriately formatted string is used to set the date and time of the newly created object.</li> <li><b>new Date(year, month, day, hour, minute, second, millisecond)</b> In the fourth form between two and seven numbers are given to specify the date and time, attribute by attribute (see under Properties).</li> </ol>
<i>Date</i> Properties	<p>There are no properties that can be get or set directly. All attributes of a <i>Date</i> object must be accessed via its methods. The <i>Date</i> attributes are:</p> <p><b>year</b> — a four digit number</p> <p><b>month</b> — an integer between 0, for January, and 11, for December</p> <p><b>day</b> — an integer between 1 and 31 specifying the day of the month</p> <p><b>hour</b> — an integer between 0, for midnight, and 23, for 11pm</p> <p><b>minute</b> — an integer between 0 and 59 that specifies the minute in the hour</p> <p><b>second</b> — an integer between 0 and 59 that specifies the second in the hour</p>

	<p><b>millisecond</b> — an integer between 0 and 999 that specified the millisecond</p>
<i>Date Methods</i>	<p><b>getDate()</b> — returns the day attribute, i.e. the day of the month between 1 and 31</p> <p><b>getDay()</b> — returns the day of the week, 0 for Sunday, 1 for Monday, ..., 6 for Saturday</p> <p><b>getFullYear()</b> — returns the year attribute (<code>getYear()</code> is deprecated version)</p> <p><b>getHours()</b> — returns the hour attribute (note plural in method name)</p> <p><b>getMilliseconds()</b> — returns the millisecond attribute (note plural in method name)</p> <p><b>getMinutes()</b> — returns the minute attribute (note plural in method name)</p> <p><b>getMonth()</b> — returns the month attribute</p> <p><b>getSeconds()</b> — returns the second attribute (note plural in method name)</p> <p><b>getTime()</b> — returns the date as the number of milliseconds since midnight, 1 January 1970</p> <p><b>getTimezoneOffset()</b> — returns the time zone difference in minutes between date and GMT</p> <p><b>getUTCDate()</b> — returns in universal time the day attribute, i.e. the day of the month between 1 and 31</p> <p><b>getUTCDay()</b> — returns in universal time the day of the week, 0 for Sunday, 1 for Monday, ..., 6 for Saturday</p> <p><b>getUTCFullYear()</b> — returns in universal time the year attribute</p> <p><b>getUTCHours()</b> — returns in universal time the hour attribute (note plural in method name)</p> <p><b>getUTCMilliseconds()</b> — returns in universal time the millisecond attribute (note plural in method name)</p> <p><b>getUTCMinutes()</b> — returns in universal time the minute attribute (note plural in method name)</p> <p><b>getUTCMonth()</b> — returns in universal time the month attribute</p> <p><b>getUTCSeconds()</b> — returns in universal time the second attribute (note plural in method name)</p> <p><b>setDate(&lt;number 1-31&gt;)</b> — sets the day attribute, i.e. the day of the month between 1 and 31</p> <p><b>setDay(&lt;number 0-6&gt;)</b> — sets the day of the week, 0 for Sunday, 1 for Monday, ..., 6 for Saturday</p> <p><b>setFullYear(&lt;four-digit number&gt;)</b> — sets the year attribute (<code>setYear()</code> is deprecated version)</p> <p><b>setHours(&lt;number 0-23&gt;)</b> — sets the hour attribute</p>

(note plural in method name)

**setMilliseconds(<number 0-999>)** — sets the millisecond attribute (note plural in method name)

**setMinutes(<number 0-59>)** — sets the minute attribute (note plural in method name)

**setMonth(<number 0-11>)** — sets the month attribute

**setSeconds(<number 0-59>)** — sets the second attribute (note plural in method name)

**setTime(<non-negative number>)** — sets the date as the number of milliseconds since midnight, 1 January 1970

**setUTCDate(<number 1-31>)** — sets in universal time the day attribute, i.e. the day of the month between 1 and 31

**SetUTCDay(<number 0-6>)** — sets in universal time the day of the week, 0 for Sunday, 1 for Monday, ..., 6 for Saturday

**setUTCFullYear(<four-digit number>)** — sets in universal time the year attribute

**setUTCHours(<number 0-23>)** — sets in universal time the hour attribute (note plural in method name)

**setUTCMilliseconds(<number 0-999>)** — sets in universal time the millisecond attribute (note plural in method name)

**setUTCMinutes(<number 0-59>)** — sets in universal time the minute attribute (note plural in method name)

**setUTCMonth(<number 0-11>)** — sets in universal time the month attribute

**setUTCSeconds(<number 0-59>)** — sets in universal time the second attribute (note plural in method name)

**toGMTString()** — returns a string representation of the date using GMT, e.g. in Internet Explorer 5: "Sun, 5 Sep 1999 17:21:48 UTC"

**toLocaleString()** — returns a string representation of the date using local time zone, e.g. in Internet Explorer 5: "09/05/1999 18:21:48"

**toString()** — returns a string representation of the date, e.g. in Internet Explorer 5: "Sun Sep 5 18:21:48 UTC+0100 1999"

**toUTCString()** — returns a string representation of the date using universal time, e.g. in Internet Explorer 5: "Sun, 5 Sep 1999 17:21:48 UTC"

**value Of()** — returns the date as the number of milliseconds since midnight, 1 January 1970

### Activity 7: Changing the window status

When you move the mouse pointer over a hyperlink, the status area in the bottom left of the browser window normally changes to reveal the link's URL. User often use this information to decide whether to follow the link. You could, instead, arrange for the status area to display some marketing information by setting the *status* property of the *window* object.

Write a HTML/JavaScript document that includes an ordinary anchor containing an unwelcome URL and a JavaScript script that assigns a more enticing string to the status portion (e.g. Sale: all USB devices at 50% discount).

You can find a discussion of this activity at the end of the unit.

### Activity 8: Semicolons to end statements

We have been using the practice of ending JavaScript statements with line breaks. Strictly speaking, a statement is terminated by a semicolon (;). When followed by a new line, the semicolon can be omitted. The following statements take advantage of this rule.

```
document.bgColor = "blue" document.fgColor = "white"
document.write("This text should be white on a blue
background.")
```

Rewrite the above statements to use semicolons.

You can find a discussion of this activity at the end of the unit.

### Activity 9: including separate JavaScript files

The <SCRIPT> tag may have an optional SRC attribute to specify the URL of a text file containing JavaScript statements that are to follow the tag. Rewrite the sample solution given for Activity 2 using this facility. (Hint: use a file in the same directory as the HTML file.)

Note that by convention an extension of .js is used as a suffix to the name of the JavaScript file, e.g. *foreColour.js*.

You can find a discussion of this activity at the end of the unit.

### Activity 9: Opening a new Window

Check what happens when you open a new window with the following JavaScript. Try to predict what will happen before trying this. In particular, try to predict what object the variable *document* refer to.

```
window.open() document.bgColor="blue" document.fgColor =
"yellow"
document.write('This is the new document')
```

You can find a discussion of this activity at the end of the unit.

Now do Review Questions 10 and 11.

## 11.4 Review Questions

### 11.4.1 Review Question 1

What is the main advantage of JavaScript?

You can find the answer to this question at the end of the unit.

## 11.4.2 Review Question 2

What is the JavaScript term for the parts of an object's state, and what is the term for a function that is a part of an object?

You can find the answer to this question at the end of the unit.

## 11.4.3 Review Question 3

Write down in your own words what happens when an interactive HTML/JavaScript document is loaded by a browser from a Web server.

You can find the answer to this question at the end of the unit.

## 11.4.4 Review Question 4

What object does the method *alert* belong to? What does the method do in that object? You can find the answer to this question at the end of the unit.

## 11.4.5 Review Question 5

Explain what an argument is and give an example of one. You can find the answer to this question at the end of the unit.

## 11.4.6 Review Question 6

Write a statement in JavaScript that displays a dialogue box with the greeting 'Form ready for your application. Please proceed.'

You can find the answer to this question at the end of the unit.

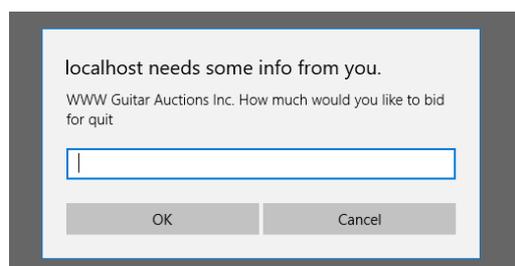
## 11.4.7 Review Question 7

What is the purpose of a variable? How do you declare a variable in one script in a document for it to be used by another?

You can find the answer to this question at the end of the unit.

## 11.4.8 Review Question 8

Write the JavaScript code that will produce the following prompt:



You can find the answer to this question at the end of the unit.

## 11.4.9 Review Question 9

The date Fri Jan 4 11:38:00 UTC 1991 is 662989080980 milliseconds after midnight, 1 January 1970. How would you create a *Date* object referred to by variable *theDate*, which represented the date Fri Jan 4 11:38:00 1991 in

universal time.

You can find the answer to this question at the end of the unit.

## 11.4.10 Review Question 10

To which object does the method `close()` belong in the following line of JavaScript?

```
close()
```

You can find the answer to this question at the end of the unit.

## 11.4.11 Review Question 11

The window object includes a method `resize` that takes two arguments which specify the width and height of the window in pixels. Read the following script and write down what it does. (Do not try to work out what happens in 'odd' situations, like when a negative number is given.)

```
var squareSize
= window.prompt("In what size square would you like this?",
"300") window.resizeTo(squareSize,squareSize)
```

You can find the answer to this question at the end of the unit.

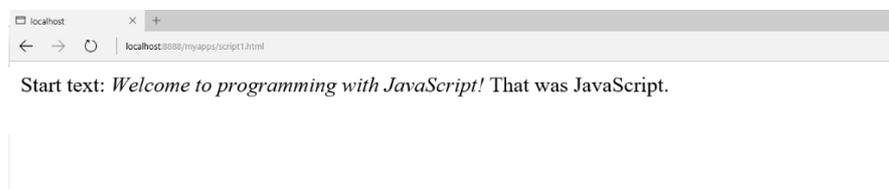
# 11.5 Discussions and Answers

## 11.5.1 Discussion of Exercise 1

It really does not matter what you display just before and just after the welcome greeting. The following HTML and JavaScript is the same as the original but with *Start text:* before and *That was JavaScript.* after. Without the document's preamble and closing tags we have:

```
Start text:
<SCRIPT>
document.write("<em>Welcome to programming with
JavaScript!</em>")
</SCRIPT>
That was JavaScript.
```

This result when run in Microsoft Edge is:



Can you explain the spacing? That is, can you see why there is no space between the '!' and the 'W', and why there is no space between the '!' and the 'T'?

The answer is that the `<SCRIPT>` and `</SCRIPT>` tags do not themselves affect the HTML stream. If they, and what they enclose, had not been there, there would be no space between the '!' and the 'T'. If you were to put the tags on separate lines, then space would have been introduced by the line breaks. Of course, you could have included spaces or other spacing tags in the argument to write — like `<BR>`.

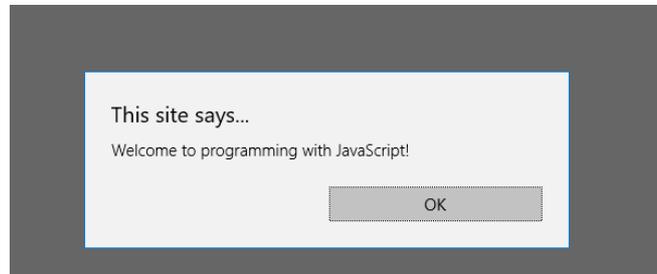
## 11.5.2 Discussion of Exercise 2

## JavaScript 1: Basic Scripting

Instead of using the `write` method with the greeting as parameter, use the `alert` method with the greeting as parameter, thus:

```
<SCRIPT>
window.alert("Welcome to programming with JavaScript!")
</SCRIPT>
```

This would produce a dialogue box similar to the one below.



You can see that the `<em>` and `</em>` tags have been omitted. The `alert` method (of the object `window`) does not get the browser to interpret its argument as HTML, so the tags would appear as ordinary text in the dialogue box if we had included them.

Note that different versions of JavaScript implement their facilities a little different from each other. This was run in Microsoft Edge.

### 11.5.3 Discussion of Exercise 3

Given that the `window` object is the one that provides for the alert facility, it is reasonable to assume that it will supply the similar facility of prompting the user. Indeed, this is the case: the method `prompt` is provided. It requires two arguments — a string to prompt the user with an explanation inside the dialogue box, and a second one to act as default input.

### 11.5.4 Discussion of Exercise 4

Since the variable `input` still refers to what data the user gave in response to the prompt on email, it can be used in a subsequent script in the same document, thus:

```
<SCRIPT>
window.alert("Email with subject '" + input + "' has been
sent.")
</SCRIPT>
```

There are several points to note about this script. First, is that there is no need to redeclare the variable `input`; redeclaring it would have had no effect. Second is the use of single quote marks within the double quotes that are used here to delimit strings. In fact, either can be used as delimiters, allowing you to use whatever is not a delimiter inside the string. Third as far as the JavaScript interpreter is concerned, the spaces either side of the concatenation operator (+) are irrelevant; they are there to help the human reader and are ignored by the interpreter.

### 11.5.5 Activity 2: Checking and Setting Background Colour

```
<!DOCTYPE html>
<!-- Copyright (c) 2015 UCT -->
<html>
<HEAD>
<TITLE>
Welcome message
</TITLE>
</head>
```

## JavaScript 1: Basic Scripting

```
<BODY>
<SCRIPT>
window.alert("background colour is (in Hex): " +
document.bgColor)
window.alert("changing colour to blue")
document.bgColor = "blue"
window.alert("background colour is (in Hex): " +
document.bgColor)
window.alert("changing colour to coral")
document.bgColor = "FF7F50"
window.alert("background colour is (in Hex): " +
document.bgColor)
</SCRIPT>
</BODY>
</html>
```

### 11.5.6 Activity 3: Setting a document's foreground colour

The following will change background colour and foreground colour as required and write some test text.

```
<!DOCTYPE html>
<!-- Copyright (c) 2015 UCT -->
<html>
<HEAD>
<TITLE>
Foreground color changes
</TITLE>
</head>

<BODY>
<SCRIPT>
document.bgColor = "blue"
document.fgColor = "white"
document.write("This text should be white on a blue background.")

</SCRIPT>
</BODY>
</html>
```

This would produce the following output.



### 11.5.7 Activity 4: Using user input to set colours

The following JavaScript does what is required. In this version, default colours are included in the prompt. These are not necessary.

```
var fore, back
back = window.prompt("Background colour:", "coral")
document.bgColor = back
fore = window.prompt("Foreground colour:", "blue")
document.fgColor = fore
document.write("This text should be "+ fore + " on a "
```

```
+ back + " background" )
```

Note the use of the variables *fore* and *back* for the foreground colour and background colour respectively. These are declared in a single *var* statement (you could have used two). Each is used to remember the data input via the *prompt* method and is subsequently used for two purposes: first, to set the associated property, and second, to form the sample text that allows you to confirm you have programmed the correct behaviour.

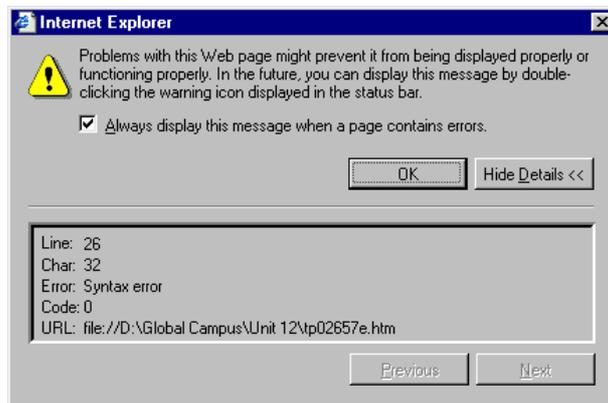
Supplying valid colours (e.g. green and yellow) other than any defaults you supply are sensible test cases. However, you must check what happens with not-so-sensible cases. For example, if you input both colours to be the same, you will not see the sample text. (However, if you use Select All from an appropriate browser menu, you should see the sample text.) If you input nonsense for either colour (e.g. %^RTkz) the variables will be assigned the nonsense value, but the properties will be changed to HEX 000000 which is the colour black.

Something quite different happens if you click Cancel on the prompting dialogue boxes. When Cancel is clicked the prompt method returns the special value *null* — essentially meaning 'no value'. If *null* is assigned to either of the colour properties, they are left unchanged (although the variables still have the value *null*). This behaviour is not easy to determine.

There is a general lesson to be learnt here: the variables you use may be set to something that does not make sense for their subsequent use and the values assigned to properties may not be the values they are given. Test thoroughly! And keep your test cases and notes of results in case you have to change your system.

## 11.5.8 Activity 5: Dealing with errors

The first thing to look at is what the script is supposed to do. The last statement in which a string is to be written into an HTML stream would suggest that some input (represented by the variable *input*) is to output underlined (you can recognise the `<U>` and `</U>` tags). In fact that is where the first problem is. Internet Explorer produces the following. In fact it does this before it executes the beginning of the script. (From this you can infer that the Microsoft interpreter does some checking of the entire script before interpreting any of it.)



In fact Netscape Navigator does not complain at all but allows the errors in the statement and sorts it out! The browser Opera 3.60 does not complain initially, but only displays the first string: 'You typed: '. Whatever the rights or wrongs of these approaches, the statement is hard to read and causes a problem for at least one browser, so should be corrected. The problem is that the underline tag `<U>` has not been enclosed in either single or double quotes. Furthermore (and no browser could complain) the *input* variable is inside the last string, and so is correctly interpreted as the sequence of characters *i n p u t*. Fixing the last line gives:

```
document.write("You typed: <U>" + input + "</U>")
```

The errors in the last line were syntactic errors — errors to do with the form (syntax) of the script. Other errors are semantic, where you write something which JavaScript cannot make sense of, or logical, where you simply do the wrong thing for the application you are trying to develop.

Working from the top of the script in Activity 4: line 1 declares *fore* and *back*, and line 2 assigns the

## JavaScript 1: Basic Scripting

document's background colour to *back*. However, *back* is not used in the rest of the script. It could be used in another script in the same document, but it could be declared and initialised to the background colour in such as script. So, let's remove *back* from the variable declaration and remove the second line:

```
var fore
```

Line 3 is a mess. It has two assignments in it. You might not think so, but this is syntactically OK. What it means is that the rightmost assignment takes place first (*document.fgColor = fore*) and then its result is assigned to the next variable to the left — to *fore*. This type of assignment can make sense, but not here, because *fore* has no value — it is *undefined* (different to *null*) — and so its assignment to *document.fgColor* attempts to make that property undefined. But it stays as it was before. To figure this out you would have needed to insert `alert` or `write` method calls before and after the multiple assignment. Not only does line 3 contain a semantic error — an inadvertent JavaScript error — it contains a logical error which is not needed for the script to fulfil its purpose. The logical error is trying to change the foreground colour, whereas it looks like *fore* is to be used in a subsequent prompt, so it does need to be set to the foreground colour. Hence, we simplify line 3 to be:

```
fore = document.fgColor
```

Line 4 seems straightforward: it appears to be prompting the user for a new foreground colour, using the current foreground colour (in *fore*) as a default (albeit as HEX, rather than as a user-friendly name). But, *newFore* has not been declared. In fact, this is legal in JavaScript, but is not helpful to the human reader and so should be included in line 1. The same is true of `input` subsequently, so let's add it to line 1 as well:

```
var fore, newFore, input
```

Line 5 assigns the object remembered by *newFore* to the document's foreground colour. However, nonsense could have been input, or Cancel may have been clicked. Either way, *newFore* may not represent the foreground colour. Either reset *newFore* to be the background colour prior to the prompt (leaving the latter as it stands). Alternatively used *document.bgColor* in both lines and remove *newFore*. Let's do the former and complete the repaired script:

```
var fore, newFore,
input fore = document.fgColor
newFore = window.prompt("Foreground colour:", fore)
document.fgColor = newFore
newFore = document.fgColor
input = window.prompt("What do you want written out
underlined in colour document.write("You typed: <U>" +
input + "</U>")
```

As you might have realised, there is nothing wrong with enclosing the final question mark in single quotes.

Note how long this activity took. Although the script was simple, a lot of thinking was needed to work out what was wrong with it. Be aware that you may have to spend a lot of time 'debugging' scripts like this.

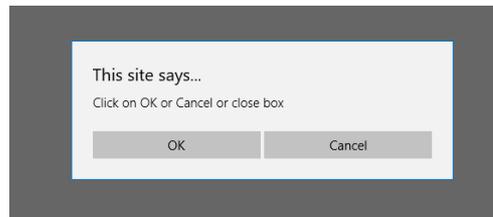
### 11.5.9 Activity 6: The confirm method

All you need is to try out the method three times — one for each of the ways in which you can complete interacting with it: clicking OK, clicking Cancel, and closing its window. For example:

```
var confirmation
confirmation = window.confirm("Click on OK or Cancel or
close box") document.write("You clicked <B>" + confirmation
+ "</B>")
```

With Microsoft Edge this produces the following dialogue box:

## JavaScript 1: Basic Scripting



Clicking OK assigns the value *true* to the variable *confirmation*, as verified in the document, below. Clicking on Cancel or closing the window of the dialog box, returns *false*. You will use *true* and *false* frequently in JavaScript, but not in this unit!

### 11.5.10 Activity 7: Changing the window status

Here is the entire body of the document:

```
<BODY>
<A HREF = "http://www.most-expensive-sellers.com" > Come to
out cheap on-line store
</A>
<SCRIPT>
window.status = 'Sale: all USB devices at 50% discount'
</SCRIPT>
</BODY>
```

Ideally, if we are to change the status text, we want to vary it as the user moves around the window. We will see how to do that when we discuss events in the next unit.

### 11.5.11 Activity 8: Semicolons to end statements

```
document.bgColor = "blue"; document.fgColor = "white";
document.write("This text should be white on a blue background.");
```

### 11.5.12 Activity 9: including separate JavaScript files

The HTML file would contain all of the HTML of the earlier version, but with the `<SCRIPT>` tag changed:

```
<!DOCTYPE html>
<!-- Copyright (c) 2015 UCT -->
<html>
<BODY>
<SCRIPT SRC="foreColour.js">
</SCRIPT>
</BODY>

</html>
```

The file `foreColour.js` would then contain:

```
document.bgColor = "blue" document.fgColor = "white"
document.write("This text should be white on a blue
background.")
```

One advantage of this style is that if a user was to save the source of a Web document, the included JavaScript file is not saved.

### 11.5.13 Activity 10: Opening a new Window

The new window is placed in the foreground and old document (with the URL of the file you use) is left in the background. The variable *document* still refers to the original document, now in the background window, hence that document has a blue background and contains new yellow text.

## 11.5.14 Answer to Review Question 1

JavaScript allows Web pages to be interactive, i.e. to part of a Web application.

## 11.5.15 Answer to Review Question 2

In the JavaScript object model, parts of an object's state are called 'properties'. A function that belongs to an object is called a method.

## 11.5.16 Answer to Review Question 3

When a user, for example, clicks on a hyperlink, the browser requests a document from a Web server. The server delivers it back to the user via his or her browser. The browser replaces the content of the current window with the newly rendered view of the document, that is, with the original document's HTML formatted. If the browser encounters JavaScript code, the instructions contained in the script are acted on by the browser.

## 11.5.17 Answer to Review Question 4

The JavaScript object *window* has methods *alert*. The *alert* method simply displays a string in a dialogue box and waits for the user to click OK (or close the dialogue box).

## 11.5.18 Answer to Review Question 5

An argument is a piece of information needed by a method. You supply an argument by enclosing it in parentheses, as in

```
document.write("<b>Delivery date: </b>")
```

, in which case the argument is

```
"<b>Delivery date: </b>".
```

## 11.5.19 Answer to Review Question 6

The following JavaScript code will display the string as required:

```
window.alert("Form ready for your application. Please proceed.")
```

## 11.5.20 Answer to Review Question 7

A variable is used to remember the result of evaluating some expression, typically to remember the result returned by a method.

A variable declared in one script in a document is available for use in all other scripts in the document.

## 11.5.21 Answer to Review Question 8

The following script would produce the required prompt.

```
window.prompt("WWW Guitar Auctions Inc. How much would you like to  
bid for guit
```

However, in reality some variables would be set to specific value and then used for in a more general prompt. For example, the company name may be held by a variable, the number of the item being sold may be held by a variable, and the minimum to be bid may be held:

## JavaScript 1: Basic Scripting

```
var company = "WWW Guitar Auctions Inc."

var itemNo; minimumBid = 1
itemNo = 2873 //in reality something more complicated would set
this minimumBid = 100 //and this
window.prompt(company+ " How much would you like to bid for
guitar #"+itemN
```

### 11.5.22 Answer to Review Question 9

You would use the following:

```
theDate = new Date(662989080980)
```

### 11.5.23 Answer to Review Question 10

Because no object name precedes it, this method must belong to the window object, and so could be written *window.close()*.

### 11.5.24 Answer to Review Question 11

The script resizes the window containing the document of which the script is part. It resizes to a square (hence the variable name *squareSize*). The script begins by prompting the user for the size of the square and suggests a size of 300.

By the way the 'odd' situations give rise to unpredictable behaviour in the browsers. Clicking Cancel sets *squareSize* to null and so the browser window may become very small, or remain unchanged — depending on the browser. The same happens with a negative number.

# Chapter 13. JavaScript 2: Event Handling

## Table of Contents

Objectives .....	2
13.1 Introduction .....	2
13.1.1 Event-based Programming .....	2
13.1.2 Event Handlers 'One Liners' .....	2
13.1.3 Events and objects .....	3
13.1.4 Anchor Events .....	4
13.2 Animating Button Images .....	7
13.3 Conditional Execution .....	9
13.3.1 JavaScript if statement .....	9
13.4 Code blocks .....	10
13.5 Boolean operators .....	11
13.6 General Selection .....	12
13.7 HTML Attributes for Event handling .....	13
13.8 Extension .....	14
13.8.1 Variables and their Scope .....	14
13.9 Review Questions .....	15
13.9.1 Review Question 1 .....	15
13.9.2 Review Question 2 .....	15
13.9.3 Review Question 3 .....	15
13.9.4 Review Question 4 .....	15
13.9.5 Review Question 5 .....	15
13.9.6 Review Question 6 .....	15
13.9.7 Review Question 7 .....	17
13.9.8 Review Question 8 .....	17
13.9.9 Review Question 9 .....	17
13.9.10 Review Question 10 .....	17
13.10 Discussions and Answers .....	17
13.10.1 Discussion of Exercise 1 .....	17
13.10.2 Discussion of Exercise 2 .....	17
13.10.3 Discussion of Exercise 3 .....	18
13.10.4 Discussion of Exercise 4 .....	18
13.10.5 Discussion of Exercise 5 .....	18
13.10.6 Discussion of Exercise 6 .....	19
13.10.7 Discussion of Exercise 7 .....	20
13.10.8 Discussion of Exercise 8 .....	20
13.10.9 Discussion of Exercise 9 .....	21
13.10.10 Discussion of Activity 1 .....	21
13.10.11 Discussion of Activity 2 .....	21
13.10.12 Discussion of Activity 3 .....	21
13.10.13 Discussion of Activity 4 .....	21
13.10.14 Discussion of Activity 5 .....	22
13.10.15 Discussion of Activity 6 .....	22
13.10.16 Answer to Review Question 1 .....	22
13.10.17 Answer to Review Question 2 .....	22
13.10.18 Answer to Review Question 3 .....	22
13.10.19 Answer to Review Question 4 .....	22
13.10.20 Answer to Review Question 5 .....	22
13.10.21 Answer to Review Question 6 .....	22
13.10.22 Answer to Review Question 7 .....	23
13.10.23 Answer to Review Question 8 .....	23
13.10.24 Answer to Review Question 9 .....	23
13.10.25 Answer to Review Question 10 .....	23

Answer to Review Question 9 .....	26
Answer to Review Question 10 .....	26

## Objectives

At the end of this chapter you will be able to:

- Write HTML files using JavaScript event handlers;
- Write HTML files using conditional statements and code blocks.

## 13.1 Introduction

The interesting behaviour of a system tends to be dependent on changes to the state of the system as a whole, or to its components. The kind of interaction a Web application might include usually involves short-term changes of state in which it is only important to know that they have occurred. That is the change of state is not intended to persist; it happens and it is not stored explicitly in the system. Such a change is indicated by an *event*. In the context of JavaScript, an event is an action that occurs in a browser that JavaScript provides facilities to detect and so act upon. Events are generally related to user interactions with the document, such as clicking and pointing the mouse, although some are related to changes occurring in the document itself. Programming JavaScript to handle such events provides for many styles of human-computer interaction. In short, programming JavaScript event handlers is crucial if you want interactive Web pages. When this style of programming dominates your design, it is known as *event-based programming*.

### 13.1.1 Event-based Programming

One event that you already know about occurs when the mouse is clicked on something, such as a hypertext link. Of course, the browser itself may intercept these events. You will note that many browsers change the status bar when the mouse is moved over an anchor. It is usually changed to the anchor's URL. In this case the browser has intercepted the event and has caused some action to occur. Events are useful for seeing what the user is doing and to provide them with extra information concerning their action.

Events are frequently used on forms to make it easier for the user to type in correct information, and to warn them when they input something incorrectly. For instance, if a text box requires a phone number, you can use events to notice whenever the user inputs data into the text box, and to ensure that the inputted data contains only numbers and dashes. Finally, you can validate all of the input before the user submits the form.

Events don't only have to be used to process forms. They could, for instance, be used when you have a number of frames which need to have their content changed when a user clicks on an anchor.

### 13.1.2 Event Handlers 'One Liners'

It is possible to add JavaScript to individual HTML tags themselves without using SCRIPT tags. These are often only single lines of code, and are thus nicknamed 'one liners'. This is not the only way to program event handlers, but is often the most convenient. This style of handling events is evidence of the close relationship between HTML and JavaScript: for a whole range of HTML elements tag attributes are provided that are associated with events. These attributes have as their value JavaScript code that is executed if the event occurs. For example, anchor tags support the event of a mouse pointer being moved over the anchor using the attribute *onMouseOver*. If a tag supports the event represented by the attribute, and the event occurs, then the JavaScript that is the value of the attribute is executed.

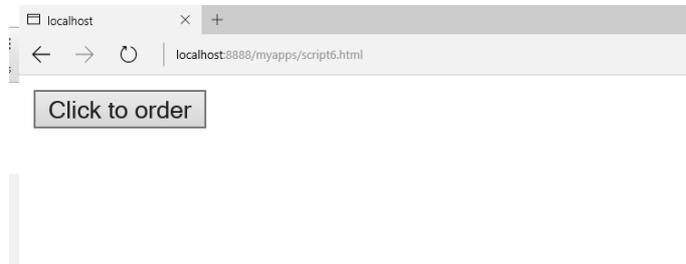
Many events can occur while a user is interacting with a Web page. For example a user might click on a button, change some text, move the mouse pointer over a hyperlink or away from one, and, of course, cause a document to load. There are event handlers for these events with names that include: *onClick*, *onMouseOver*, *onMouseOut*, *onLoad*. (We will be making use of all of these later.)

One of the simplest events is a mouse click. It is represented by the attribute *onClick* and supported by links and HTML button elements. Examine the following tag with an event handler as an attribute — a so-called 'one-liner'. (We will assume that this tag appears between *<FORM>* and *</FORM>* tags.)

## JavaScript 2: Event Handling

```
<INPUT type="button" value="Click to order"
onClick="window.alert('Purchase') ">
```

When a browser interprets this tag, it **renders** a button labelled *Click to order*. Subsequently, if a user clicks on that button, the click event is detected and the JavaScript associated with its attribute is executed. Here, an alert dialogue box is displayed, as shown below.



Let us work through this HTML and JavaScript. The first part of the `<INPUT>` tag is as you have previously seen: the *type* attribute is assigned the value *button*; the *value* attribute, which labels the button, is assigned the value *Click to order*. Then comes the new attribute for the tag `<INPUT>`. It is *onClick*, and is given the value that in this case is a single JavaScript statement that invokes the `window.alert()` method. This final attribute assignment creates an event handler for the particular JavaScript object representing the button such that clicking on the visual representation of the button causes the code to be executed.

In general, a sequence of statements may be included in the event handler. However, as it is essentially a 'one-liner'. Each line in the sequence must be separated by semicolons (as would be done in Java).

For example, including a second dialogue box that said 'Have a nice day' would require a semicolon, as in:

```
<INPUT type=button value="Click to order"
onClick="window.alert('Purchase window.alert('Have a nice
day') ">
```

This HTML/JavaScript works just as previously, except that clicking on the 'Click to order' button will produce a second alert box after the first has been dismissed.

### Exercise 1

Modify the earlier *onClick* example to include a flashing background colour before the alert dialogue box. Make sure you restore the initial background colour by saving it first with a variable and using the variable to restore the colour at the end. Depending on the speed of your computer, you will probably need at least two colour changes to notice anything.

You can find a discussion of this exercise at the end of the unit.

## 13.1.3 Events and objects

Earlier, it was suggested that you could conceive of the button as being an object with a nameless method that is invoked when you click on the button visible via a browser. You can, in fact, see that the HTML tag is implicitly creating a button object by accessing the button object and its properties - its type, as defined in the HTML tag, and its value, the text shown as the button label and defined by the VALUE tag. To do so the special variable *this* is used to refer to the object which the method belongs to. Hence, *this.type* can be used to access the type property, and *this.value* can be used to access the value property. The following variation of the first event handler can be used to confirm the object nature of the HTML button element:

```
<INPUT type=button name="orderButton" value="Click to order" onClick="windo
this.type + ' and has value: ' + this.value) ">
```

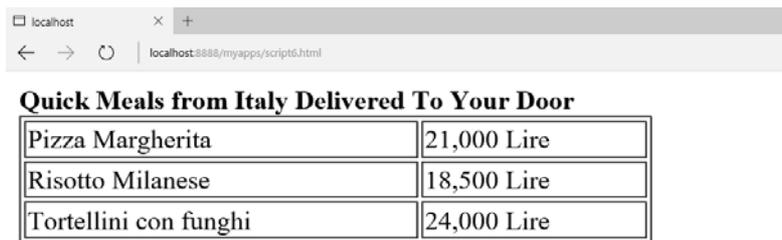
Executing this HTML `<INPUT>` tag (in a form) will produce the button as before, but when you click on it, the alert dialogue box now shows the two properties of the object referred to by *this*.

**Note**

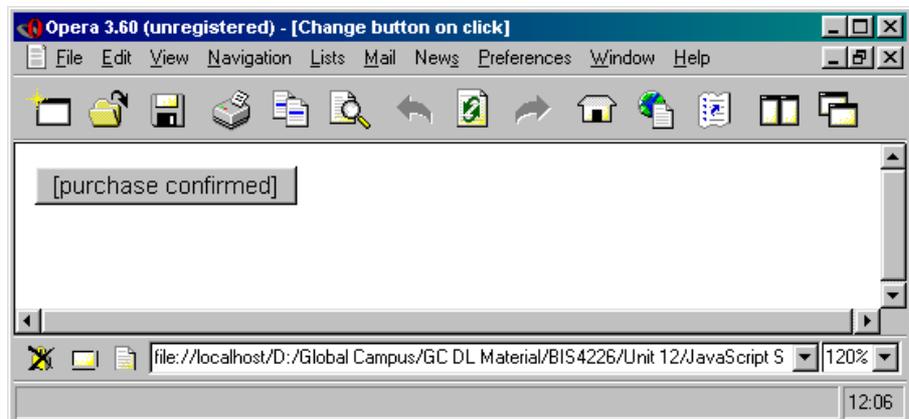
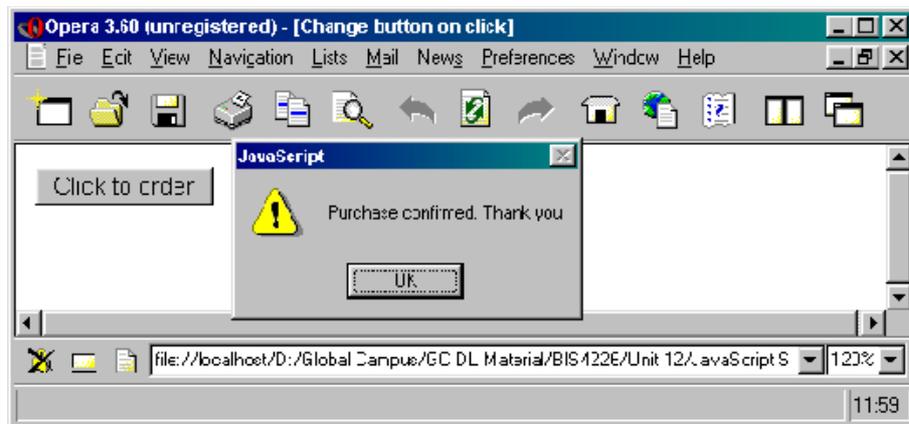
Note that you can apparently change some properties of such an object. It is not clear that you would ever need to change the type of a button (e.g. from this sort of action button to a radio button) but you might want to change the label.

**Exercise 2**

Examine the original *onClick* example, which confirms a purchase, and the previous variation in which the button properties are accessed via the *this* keyword. Then devise HTML/ JavaScript that confirms a purchase, as in the original example of *onClick*, but which changes the label of the button after the confirmation to *[Purchase confirmed]*. See the diagrams below for the sort of thing you are aiming for.



Hint: use the *this* keyword like a variable to assign a new string to the *value* property so that the new string is the text on the button.



You can find a discussion of this exercise at the end of the unit.

**13.1.4 Anchor Events**

As mentioned earlier, the anchor tag `<A>` can be enhanced to include JavaScript event handlers

## JavaScript 2: Event Handling

that correspond to the mouse pointer moving over the enclosed link, moving away from it and clicking on it. These anchor tag attributes are, respectively, *onMouseOver*, *onMouseOut* and *onClick*.

The general form is similar to that for *<INPUT>* with the event attribute following the link. Say you wanted to warn a user who had clicked on a link that the URL was not necessarily what they had wanted. For example, there is a UK company whose website URL is *www.apple.co.uk*. The company is not the UK division of Apple Computer Inc., so it might help a user to warn them what the link they had clicked on was maybe not what they wanted. (After the warning the user could stop the browser connecting to the server and go back.) The HTML/JavaScript is as follows:

```
<A href="http://www.apple.co.uk" onClick="alert('Remember this is not the Apple Computer Site')">Apple.co.uk</A>
```

Of course warning a user after he or she has done something (clicked on the link) is not as helpful before one given before the action. The mouse-over event, which is programmed using the *onMouseOver* attribute allows this. For example:

```
<A href="http://www.apple.co.uk" onMouseOver="alert('Remember this is not the Apple Computer Site')">Apple.co.uk</A>
```

This tag differs from the previous one only by the replacement of *onClick* by *onMouseOver*. When the user moves the mouse pointer over the link, the dialogue box appears with its warning. Thus, the user can avoid the URL.

However, even this style is not optimal for the user. The warning can interfere with the interaction, requiring to be dismissed by clicking on OK or pressing the Enter key. A common practice is to use the window's status area, just as we did in the previous unit. We can avoid a browser's default behaviour of displaying a URL in the status area of the window's bottom bar, by inserting something more helpful to the purposes of the document — such as the kind of warning just discussed.

Let us take a different example, in which a document is meant to sell something to its user. You might to encourage the user to follow a link to some on-line shopping as soon as he or she moves over the link to the on-line shop, as in the following. First the document provides some ordinary text, followed by a link that reads ". Placing the mouse pointer over the link generates the text in the status area.



Note that the text 'Click here to get to the bargains!' only appears in the status area when the mouse pointer is over the link. This is achieved using the mouse-over event. As the name suggests, when the mouse pointer is over the link, the appropriate JavaScript code is executed. Here is what produces this interaction:

```
<P>
There's a sale on. Come to our on-line shop for lots of
bargains.
</P>

<P>
<AHREF = "http://www.most-expensive-sellers.com"
```

## JavaScript 2: Event Handling

```
onMouseOver = "window.status = 'Click here to get  
to the bargains!';return >  
</P>
```

### Exercise 3

Write down in your own words an explanation of what the above HTML and JavaScript in the anchor tag does.

You can find a discussion of this exercise at the end of the unit.

### Note

Note that the status area may not change back once the mouse pointer leaves the anchor, as this behaviour varies among browsers.

Strictly speaking there is something missing from the JavaScript of the *onMouseOver* event handler. It can be used for other actions than changing the status bar and it is useful for the browser to know whether the URL should be shown in status area (the usual behaviour) or not. For instance, in the first *onMouseOver* example the URL would still be shown in the status area after the alert dialogue box had been dismissed, especially if Enter had been used — because the mouse pointer would still be over the link. Including a *return true* statement at the end of the JavaScript 'one-liner' tells the browser that it should not display the URL. Although the status bar is to be occupied by our exhortation to come shopping, it is better to include the return value, as below:

```
<A HREF = "http://www.most-expensive-sellers.com"  
onMouseOver = "window.status = 'Click here to get to  
the bargains!'; return true;">Come to our cheap on-  
line store </A>
```

The use of a return value is a very important part of the above event handler. You will recall from our abstract object model (in Unit 10) that messages to objects can evoke a response. We encountered this with the *window.prompt* and *window.confirm* methods that return values. Many event handlers need to return a value for the browser to make use of. In the case in point, *true* is the response if we do not want the browser to display the URL in the anchor tag — exactly what is required here as we plan to change the status area anyway. However, if we want to see the URL, we must script the event handler to return *false*, as in the script below that changes the background colour but does not change the status area:

```
<A HREF = "http://www.most-expensive-sellers.com"  
onMouseOver = "document.bgColor = 'coral';  
return false">Come to our cheap on-line store</A>
```

As we have indicated, there is also an event that represents the mouse pointer leaving a link. This 'mouse-out' event is represented by the attribute *onMouseOut* and the code associated with it is executed when a user moves the mouse pointer away from a link.

### Exercise 4

Change the scripting in the previous anchor to restore background colour to what it was before being changed to coral. Hint: use a variable to remember the background colour property of the document's state. (You may find it convenient to look again at the discussion of variables in the previous unit.)

You can find a discussion of this exercise at the end of the unit.

The alternative version of the script in the previous exercise shows a common situation for which there is a shorthand notation. The situation is where a variable is declared and soon afterwards it is initialised, i.e. set to a first value, just as in the second version of the previous script.

The shorthand allows initialisation at the same point as declaration, for example:

```
<SCRIPT>
var origBgCol = document.bgColor
</SCRIPT>
```

Be careful when using variables with handlers. As a general programming rule, you should declare variables close to where you use them. This might suggest that you should declare *origBgCol* in the handlers for both events, but this does not work because of rules of JavaScript. If handlers need to use the same variable (because they need to use the value the variable holds ) then declare the variable external to the handlers. (See the extension work on this topic)

### Activity 1: Clicking on Input Buttons

1. Implement the JavaScript of Exercise 1 to ensure you are familiar with the concept of handling an event using an HTML attribute.
2. Modify your implementation to prompt the user for his or her name, and then modify the button label using the button object's value property to include that name. With our current knowledge of JavaScript the button can only be modified in the event handler, so have it change after the first click.

Remember that if your PC or video card is very fast, you will probably not see the colours flashing.

You can find a discussion of this activity at the end of the unit.

### Activity 2: Programming Anchor Events

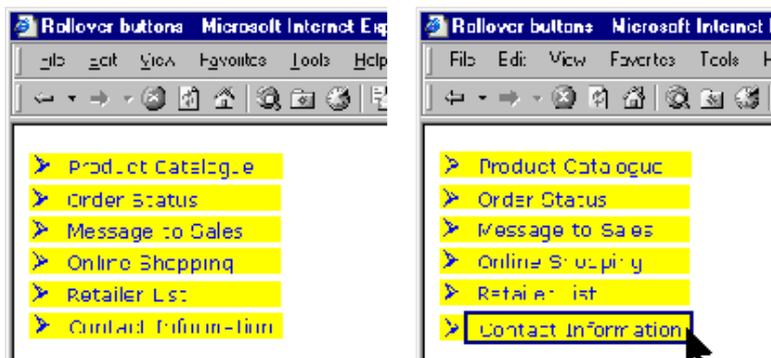
Change the HTML/JavaScript that exhorted you to go shopping for bargains (just before Exercise 4) so that when you click on the anchor the window status area changes to the greeting, 'Enjoy your shopping!' To test the JavaScript, *return false* from the handler to prevent the browser from following the link.

You can find a discussion of this activity at the end of the unit.

Now do Review Questions 1, 2 and 3

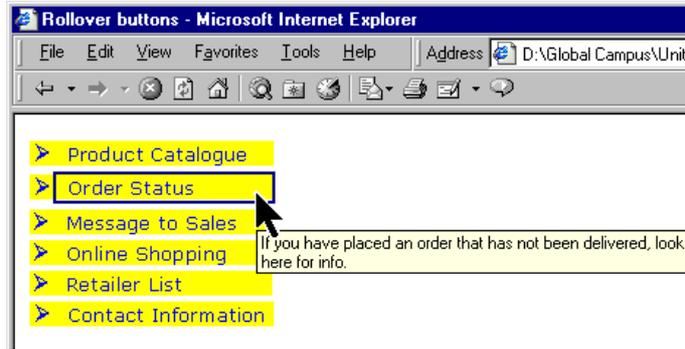
## 13.2 Animating Button Images

A common use for event handlers is to produce small animations to reinforce some aspect of the user interface. There are many situations where this may help the user to focus on some part of a Web page. For example, the buttons used in the screen shot on the left below can be enhanced by highlighting the button, as in the screen shot on the right below. This style of animation is known as a 'Rollover'.



Indeed, most rollover arrangements will also include the text value of the images' *ALT* attributes, as in the diagram below:

## JavaScript 2: Event Handling



How can you script such an interaction? The basic strategy is to swap an image that represents an unselected button, for example the first image below, with an image that represents the selected button, such as the second image below, and then swap it back.



### Exercise 5

Can you think of the event or events that would trigger these swaps? And can you guess how you would refer to the image object from within the event handlers?

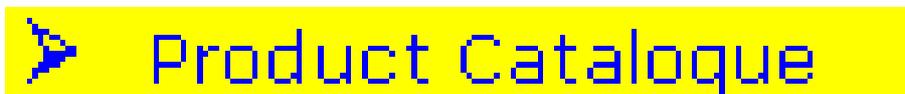
You can find a discussion of this exercise at the end of the unit.

The HTML/JavaScript code for the button rollover is as follows:

```
<IMG VSPACE=2 SRC="prod-cat-button.gif"
  onMouseOver="this.src='prod-cat-but
  onMouseOut="this.src='prod-cat-button.gif'"
  ALT="Check out the full Catalspecial orders.">
```

The *IMG* tag is just as you have previously seen. Ignoring the event handlers, it guarantees vertical spacing of two (via *VSPACE* attribute) and specifies the source of the image to be *prod-cat-button.gif* (via the *SRC* attribute); at the end of the tag it specifies the alternative (*ALT*) to the graphic image that also provides help to the user when the mouse pointer is over the button image.

As you might guess from the earlier use of the *this* variable with button objects, *this* is used in the event handlers to refer to the object corresponding to the graphical image being used as a button. As *IMG* tags have a *SRC* attribute, so to image objects have a *src* property accessible in JavaScript. Hence, within each of the event handlers the *src* property is changed to modify what image is being shown. In the mouse-over event handler, the image source is changed to refer to the graphic image that represents the selected version of the button. So, for example, if the image source specifies the file containing the first image below, moving over the image will change its source to specify the file containing the second image. When the mouse pointer leaves the image, the original graphic (image 1) is restored.





### Exercise 6

Complete the HTML/JavaScript for the button rollover interaction shown in the previous diagram using the 'this' style of referencing the image and placing break (<BR>) tags after each image (<IMG>) tag.

You can find a discussion of this exercise at the end of the unit.

### Activity 3: onMouseOver Event with an image

Choose a gif, jpg or png image of your choice. Write an HTML page that displays the image and produces a warning dialog box when the mouse pointer is over the image.

You can find a discussion of this activity at the end of the unit.

### Activity 4: onMouseOver Event with an image

Produce an HTML page that displays the image you used in Activity 3, but in this page make the status bar gives different messages according to whether or not the mouse pointer is over the image. You will need to use the event handler *onMouseOut*.

You can find a discussion of this activity at the end of the unit.

### Activity 5: changing an Image object

Every IMG HTML tag has an equivalent JavaScript image object. To make these objects easily accessible it is possible to provide a NAME attribute to the IMG tag. For example, to give an image the name "change", we would use the following IMG attribute:

```
NAME="change".
```

Now, in any JavaScript used in the HTML page, the image object will be stored in a variable called *change*. So, to access the image's src property, you could use the following code:

```
change.src
```

Now, try to set up an image and change it to another image when the mouse goes over it. If you have problems, look at the source code for the example above.

You can find a discussion of this activity at the end of the unit.

## 13.3 Conditional Execution

As in Java, JavaScript provides an *if* statement which, based on the value of a variable or other expression, can conditionally choose particular JavaScript code to execute.

### 13.3.1 JavaScript if statement

The *if* statement appears exactly as it does in Java (an expression that evaluates to *true* or *false* must be included in parentheses):

```
if (true or false expression)
statement
```

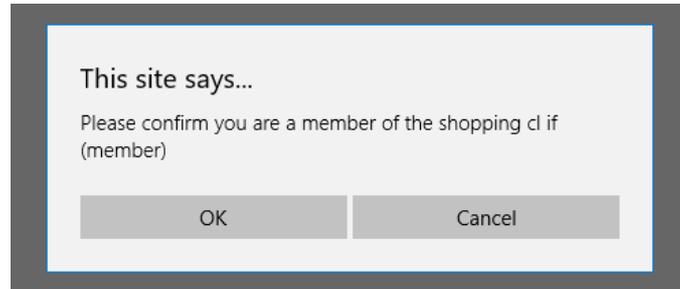
The expression in parentheses is evaluated and if it is true the following *statement* is executed; if it is false, it is not. Recall that the *window.confirm()* method returns *true* or *false* depending on

## JavaScript 2: Event Handling

whether the user clicks OK or Cancel. We will now use `confirm()` in an example of the *if* statement:

```
var member
member = window.confirm('Please confirm you are a member of
the shopping club')
if (member)
window.alert('Welcome to the Shopping Club')
</SCRIPT>
```

This code produces a confirmation dialogue box. When the user clicks OK a welcome dialogue box appears, as below.



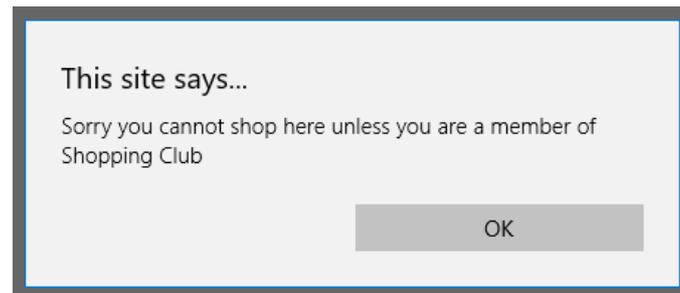
*if* statements also have a matching *else* statement, which behaves as in Java:

```
if(true or false expression)
  statement_1
else
  statement_2
```

`statement_2` is only executed if the expression in parentheses evaluates to false.

### Exercise 7

Change the previous script to display the following dialogue box if Cancel is clicked by the user when asked to confirm membership.



You can find a discussion of this exercise at the end of the unit.

## 13.4 Code blocks

To execute more than one statement in a conditional, you may use code blocks, which are sequences of statements packaged together between braces — `{` and `}`. Code blocks behave just as they do in Java, and can be used in the same places.

The most general form of an *if* statement can be written as follows:

```
if (true or false expression)
{
  statement_1a
```

```

    statement_1b
  }

```

and

```

if (true or false expression)
{
  statement_1a
  statement_1b
  statement_1c
  ...
  statement_1n
}
else
{
  statement_2a
  statement_2b
  statement_2c
  ...
  statement_2n
}

```

Let us extend the shopping example from the introduction to the *if* statement to provide the user with a table of possible purchases if they are a member of a shopping club.

### Exercise 8

Remembering that you can dynamically create the content of an HTML document using the *document.write()* method, change the shopping example to ask for confirmation of, say membership of an Italian food shopping club. If the user clicks OK, they should be presented with a table of shopping choices, as in the following:

#### Quick Meals from Italy Delivered To Your Door

Pizza Margherita	21,000 Lire
Risotto Milanese	18,500 Lire
Tortellini con funghi	24,000 Lire

You can find a discussion of this exercise at the end of the unit.

## 13.5 Boolean operators

So far we have used the *window.confirm()* method that is guaranteed to return either *true* or *false*. In a more general case, you will obtain true / false values by comparing two or more values.

Of course, you need to be able to compare all kinds of values and make a variety of comparisons. Here are the main operators for doing, which should be familiar to you from the Java module:

Operator	Example	Meaning
==	a == b	equality between any two values; returns true or false (example tests for a being equal to b)
!=	a != b	inequality between any two values; returns true or false (example tests for a not being equal to b)

Operator	Example	Meaning
===	a === b	identity between any two objects; returns true or false (example tests to see if a refers to the same object as b)
!==	a !== b	non-identity between any two objects; returns true or false (example tests to see if a does not refer to the same object as b)
<	a < b	less than between any two values; returns true or false (example tests for a being less than b)
<=	a <= b	less than or equal to between any two values; returns true or false (example tests for a being less than or equal to b)
>	a > b	greater than between any two values; returns true or false (example tests for a being greater than b)
>=	a >= b	greater than or equal to between any two values; returns true or false (example tests for a being greater than or equal to b)
&&	a && b	logical 'and' between two Boolean (true/false) values (example returns true only if both a and b are true, and false otherwise)
	a    b	logical 'or' between two Boolean (true/false) values (example returns false only if both a and b are false, and true otherwise)
!	!a	logical 'not' of one Boolean (true/false) value (example returns false if a is true and true if a is false)

## 13.6 General Selection

Because of the simple general form of the *if ... else* statement, you can use it as a very general selection mechanism by combining *if* statements in sequence, as in the code below, which determines the day of the week (in English) from the numeral for that day held by a *Date* object. Days of the week were numbered from 0 for Sunday, through to 6 for Saturday.

```
<SCRIPT>
var today = new Date()
var dayNo =
today.getDay() if (dayNo
== 0)
dayName = 'Sunday'
else if (dayNo ==
1) dayName =
'Monday' else if
(dayNo == 2)
dayName =
'Tuesday' else if
```

```

(dayNo == 3)
dayName =
'Wednesday' else
if (dayNo == 4)
dayName =
'Thursday' else if
(dayNo == 5)
dayName = 'Friday'
else if (dayNo ==
6)
dayName = 'Saturday'
window.alert('Local day
is ' + dayName)
</SCRIPT>

```

Note that there are other control statements in JavaScript. An alternative to using multiple *if...else* statements would be to use a *switch statement*. Use your knowledge of Java to experiment with these statements.

### Exercise 9

*Navigator* objects provide properties and methods for manipulating the Web browser. A pre-declared variable called *navigator* can be used for this purpose. The following code writes out details of the browser:

```

document.write('appName = ',
navigator.appCodeName, '<BR>') document.write('appName
= ', navigator.appName, '<BR>')
document.write('appVersion = ', navigator.appVersion,
'<BR>') document.write('language = ',
navigator.language, '<BR>') document.write('platform =
', navigator.platform, '<BR>')

```

You can find a discussion of this exercise at the end of the unit.

### Activity 6: Remembering an Event

Modify the code of Exercise 2 (reproduced below), in which a button changed labels after being clicked, so that clicking the new button has no effect.

```

<FORM>
<INPUT type=button value="Click to order"
onClick="window.alert('Purchase confirmed. Thank
you'); this.value = '[purchase confirmed]'">
</FORM>

```

You can find a discussion of this activity at the end of the unit.

Now do Review Questions 7, and 8.

## 13.7 HTML Attributes for Event handling

There are many HTML attributes for handling events in JavaScript. The following table lists the main ones, and the objects that support them. You do not have to memorise this list, but you should be aware of the sort of events that can be handled.

HTML Event Attribute	Object supporting event
----------------------	-------------------------

HTML Event Attribute	Object supporting event
<i>onAbort</i> — triggered when document loading interrupted	<i>Image</i>
<i>onBlur</i> — triggered when an input element loses focus	<i>Text</i> elements, <i>Window</i> , other elements
<i>onChange</i> — triggered when user selects or deselects an element or enters text and moves focus to another input element	<i>Select</i> , text input elements
<i>onClick</i> — triggered when user clicks once; return false to cancel default action of following a link, submitting, etc.	<i>Link</i> , button elements
<i>onError</i> — triggered when an error occurs while loading an image	<i>Image</i>
<i>onFocus</i> — triggered when an element is given focus	<i>Text</i> elements, <i>Window</i> , other elements
<i>onKeyDown</i> — triggered when a key is pressed down by user; return false to cancel	<i>Document</i> , <i>Image</i> , <i>Link</i> , <i>Text</i> elements
<i>onKeyPress</i> — triggered when key is either pressed or released; return false to cancel	<i>Document</i> , <i>Image</i> , <i>Link</i> , <i>Text</i> elements
<i>onKeyUp</i> — triggered when the user released a key; return false to cancel	<i>Document</i> , <i>Image</i> , <i>Link</i> , <i>Text</i> elements
<i>onLoad</i> — triggered when document or image finishes loading	<i>Window</i> , <i>Image</i>
<i>onMouseDown</i> — triggered when mouse left button pressed down by user; return false to	<i>Document</i> , <i>Image</i> , <i>Link</i> , button elements
<i>onMouseOut</i> — triggered when mouse is moved away from element	<i>Link</i> , <i>Image</i> , <i>Layer</i>
<i>onMouseOver</i> — triggered when mouse pointer moved over element; for anchors, return true to	<i>Link</i> , <i>Image</i> , <i>Layer</i>
<i>onMouseUp</i> — triggered when user releases mouse button; return false to cancel	<i>Document</i> , <i>Image</i> , <i>Link</i> , button elements
<i>onReset</i> — triggered when form reset requested; return false to stop reset	<i>Form</i>
<i>onResize</i> — triggered when window is resized	<i>Window</i>
<i>onSubmit</i> — triggered when form submission requested; return false to stop submission	<i>Form</i>

Now do Review Questions 9 and 10.

## 13.8 Extension

### 13.8.1 Variables and their Scope

As you might have discovered, if you forget to declare a variable JavaScript will implicitly declare it for you. That is, if you write `origBgCol = document.bgColor`, JavaScript will declare the variable `origBgCol` if it has not previously been declared in some way (such as with a `var` statement). However, if you attempt to retrieve a value from a variable that has not been declared, as in `document.bgColor`

`= origBgCol`, then JavaScript will report an error.

Also associated with variable declaration is the variable's scope and its search chain. These define respectively

the region of the JavaScript program in which the variable can be seen, and the way in which JavaScript looks for the variable, and operate in the same way as they do in Java. Details can be found in JavaScript, The Definitive Guide, 3rd edition, by David Flanagan, O'Reilly Books, 1998. The rules explain why the following solution to Exercise 4 will not work:

```
<AHREF = "http://www.most-expensive-sellers.com"
  onMouseOver="var origBgCol=document.bgColor;
  document.bgColor = 'coral';return false"
  onMouseOut="document.bgColor=origBgCol">
Come to our cheap on-line store
</A>
```

In the above example, the scope of *origBgCol* is the *onMouseOver* event handler, as this is where the variable is declared. Since the *onMouseOver* and *onMouseOut* scopes do not overlap, when JavaScript searches for *origBgCol* in the *onMouseOut* handler will not find the variable, and since it is attempting to obtain a value from an undeclared variable, an error is reported. Declaring *origBgCol* in the *onMouseOut* handler would not work either, as all it would do is declare two separate variables which have the same name in different scopes.

The solution to this is to declare the variable outside of both handlers, in some scope which they can both access.

## 13.9 Review Questions

### 13.9.1 Review Question 1

An event takes place and is conveyed to the Web browser and thence to the document. Describe what it means to handle an event.

You can find the answer to this question at the end of the unit.

### 13.9.2 Review Question 2

Regardless of whether a handler has been provided, when a user of a Web browser clicks the mouse an mouse-down event occurs. True or false?

You can find the answer to this question at the end of the unit.

### 13.9.3 Review Question 3

Do all events have to be captured by JavaScript code?

You can find the answer to this question at the end of the unit.

### 13.9.4 Review Question 4

When do you return true from an anchor event handler?

You can find the answer to this question at the end of the unit.

### 13.9.5 Review Question 5

What is the special keyword that acts like a variable that you use to refer to an object in its event handler?

You can find the answer to this question at the end of the unit.

### 13.9.6 Review Question 6

## JavaScript 2: Event Handling

What is the basic strategy for animating button rollovers? You can find the answer to this question at the end of the unit.

## 13.9.7 Review Question 7

What is the name of the variable that refers to the browser object? You can find the answer to this question at the end of the unit.

## 13.9.8 Review Question 8

True or false: you can't capture a mouseOver event unless the mouse is over an anchor or an image. You can find the answer to this question at the end of the unit.

## 13.9.9 Review Question 9

Which is not an event handler attribute?

1. *onMouseClicked*
2. *onMouseOut*
3. *setTimeout*
4. *onLoad*

You can find the answer to this question at the end of the unit.

## 13.9.10 Review Question 10

True or false: Events occur on objects.

You can find the answer to this question at the end of the unit.

# 13.10 Discussions and Answers

## 13.10.1 Discussion of Exercise 1

The event handler starts the same with the HTML attribute *onClick*="... Then the variable for saving the initial background colour must be declared: *var currentBack*; with a semicolon terminating the statement. Then comes the colour changes, each with semicolons at the end of the statement. Next the initial colour is restored with *document.bgColor = curentBack*. A semicolon is needed after this too. The last statement is the *alert* message to *window*, as before. Putting it all together gives:

Note to implementer: the statement below must be on one line.

```
<INPUT type=button value="Click to order" onClick="var currentBack;
currentBack = document.bgColor; document.bgColor = 'blue';
document.bgColor = 'coral'; document.bgColor = 'blue';
window.alert('Purchase confirmed. Thank you')">
```

Note that if you were to try this on a fast PCs you might find that the flashing is not noticeable.

## 13.10.2 Discussion of Exercise 2

The HTML/JavaScript code is given below.

```

<FORM>
<INPUT type=button value="Click to order"
onClick="window.alert( 'Purchase
</FORM>

```

The JavaScript works as before: when the user clicks on the button a dialogue box is displayed. Then the label is changed using the assignment *this.value = '[purchase confirmed]'* (which is, of course, preceded by a semicolon).

But, what will happen if the user were to click on the newly labelled button? Exactly the same thing would happen. The handling of the previous *onClick* event has not been remembered — events are not remembered, they continuously happen — and so the behaviour of this little system remains the same. Later we will see how an event can, in effect, be remembered and that memory used to affect behaviour.

### 13.10.3 Discussion of Exercise 3

The first part of the tag and the text that immediately precedes the closing `</A>` tag are standard HTML and together set up a link to `Apple.co.uk` with the specific URL. What follows the value for the `HREF` attribute, the URL, is an attribute that specifies in JavaScript what reaction there is to be to an event that is expected to happen with this HTML tag. The expected event is that the user will place the mouse pointer over the anchor; this is denoted by the attribute *onMouseOver*. The value of the attribute (i.e. what follows the = symbol) is the JavaScript to 'handle' the event. The JavaScript simply changes the status area of the browser's border. So, the combination of the *onMouseOver* attribute corresponding to the mouse-over event, and the JavaScript to change the window status means that when the mouse pointer is placed on this link, the message string is displayed in the browser's status bar.

### 13.10.4 Discussion of Exercise 4

The JavaScript code is given below.

```

<SCRIPT>var origBgCol</SCRIPT>

<AHREF = "http://www.most-expensive-
sellers.com"
onMouseOver="origBgCol=document.bgColor;
document.bgColor = 'coral';return false"
onMouseOut="document.bgColor=origBgCol">
Come to our cheap on-line store<A>

```

The most important aspect of this code is the use of the *origBgCol* variable to remember the original background colour. The assignment takes place as part of handling mouse-over event, while the variable is declared *outside* the event handler. Alternatively, the assignment can also be done outside the event handler, thus:

```

<SCRIPT>
var origBgCol
origBgCol=document.bgColor
</SCRIPT>

<AHREF = "http://www.most-expensive-sellers.com"
onMouseOver="document.bgColor = 'coral';return false"
onMouseOut="document.bgColor=origBgCol">
Come to our cheap on-line store<A>

```

### 13.10.5 Discussion of Exercise 5

For the rollover style of interaction you need to swap the button image with the highlighted image when the mouse-over event is triggered, and swap it back when the mouse-out event takes place.

To refer to the image object whose image is to be swapped you need to use the *this* keyword.

## 13.10.6 Discussion of Exercise 6

Your code might look something like the one below. Note how *this* is used to denote the image object within whose tag the JavaScript appears.

```
<IMG VSPACE=2 SRC="prod-cat-button.gif"
  onMouseOver="this.src='prod-cat-button-sel.gif'"
  onMouseOut="this.src='prod-cat-button.gif'"
  ALT="Check out the full Catalogue with details of
    special orders.">

<BR>

<IMG VSPACE=2 SRC="order-status-button.gif"
  onMouseOver="this.src='order-status-button-
sel.gif'" onMouseOut="this.src='order-status-
button.gif'"
  ALT="If you have placed an order that has not been
    delivered, look here for info.">

<BR>

<IMG VSPACE=2 SRC="message-button.gif"
  onMouseOver="this.src='message-button-sel.gif'"
  onMouseOut="this.src='message-button.gif'"
  ALT="Submit a form requesting sales information or
    for someone to call you.">

<BR>

<IMG VSPACE=2 SRC="online-shop-button.gif"
  onMouseOver="this.src='online-shop-button-sel.gif'"
  onMouseOut="this.src='online-shop-button.gif'"
  ALT="Registered customers can browse products and
    prices and place orders online.">

<BR>

<IMG VSPACE=2 SRC="retailers-button.gif"
  onMouseOver="this.src='retailers-button-sel.gif'"
  onMouseOut="this.src='retailers-button.gif'"
  ALT="Find out about shops near you where you can
    examine and buy our products.">

<BR>

<IMG VSPACE=2 SRC="contacts-button.gif"
  onMouseOver="this.src='contacts-button-sel.gif'"
  onMouseOut="this.src='contacts-button.gif'"
  ALT="Names, postal addresses, telephone and fax
    numbers for you to contact all our departments.">
```

```
<BR>
```

## 13.10.7 Discussion of Exercise 7

```
<SCRIPT>
var member
member = window.confirm('Please confirm you are a member of the
shopping club')
  if (member)
window.alert('Welcome to the Shopping Club')
else
window.alert('Sorry you cannot shop here unless you are a member of Shopping
Club')
</SCRIPT>
```

## 13.10.8 Discussion of Exercise 8

The following JavaScript achieves the desired behavior. Notice the form of the *if ... else* statement and how code blocks are used.

```
<!DOCTYPE html>
<!-- Copyright (c) 2015 UCT -->
<html>
<BODY>
<SCRIPT>

var member
member = window.confirm('Please confirm you are a member of the Italian food
club')

if (member)
{
  //next are JavaScript document writes to include a table
  //of items and prices document.write('<TABLE BORDER=1>')
  document.write('<CAPTION><B>Quick Meals from Italy Delivered To Your
Door</B></CAPTION>')
  document.write('<TABLE style= "width: 40%" border = "1">')
  document.write('<TR>')
  document.write('<TD>Pizza Margherita</TD><TD>21,000 Lire</TD>')
  document.write('</TR>')
  document.write('<TR>')
  document.write('<TD>Risotto Milanese</TD><TD>18,500 Lire</TD>')
  document.write('</TR>')
  document.write('<TR>')
  document.write('<TD>Tortellini con funghi</TD><TD>24,000 Lire</TD>')
  document.write('</TR>')
  document.write('</TABLE>')
}
else
{
window.alert('Sorry, please go back to the membership enrolment page')
//next is JavaScript to go back to a membership form
document.write('<A href="http://www.food-
shopping.co.it/registration/register/"> Registration form </A>')
}
</SCRIPT>
</BODY>
</html>
```

## 13.10.9 Discussion of Exercise 9

```
if (navigator.appName.substr(0,8) == 'Microsof')

window.alert('This is a Microsoft browser')
if (navigator.appName.substr(0,8) == 'Netscape')
window.alert('This is a Netscape browser')
```

## 13.10.10 Discussion of Activity 1

Clicking on Input Buttons

1. The code from Exercise 1 is repeated below:

```
<INPUT type=button value="Click to order"
  onClick="var currentBack; curentBack = document.bgColor;
  document.bgColor = 'blue'; document.bgColor = 'coral';
  document.bgColor = 'blue';; window.alert('Purchase
  confirmed. Thank you')">
```

2. The modified version is given below.

```
<FORM>
<SCRIPT>
var person = prompt('What is your name?','')
</SCRIPT>
<INPUT type=button value="Click to order"
  onClick="var currentBack; curentBack = document.bgColor;
  document.bgColor = 'blue'; document.bgColor = 'coral';
  document.bgColor = 'blue';; window.alert('Purchase
  confirmed. Thank you '+person);
  this.value='Click to confirm, '+person">
</FORM>
```

## 13.10.11 Discussion of Activity 2

```
<A HREF = "http://www.most-expensive-sellers.com"
  onClick = "status='Enjoy your shopping!';return false">
Come to our cheap on-line store</A>
```

## 13.10.12 Discussion of Activity 3

Your code might look something like:

```
<IMG SRC="right.gif" onMouseOver="window.alert('Warning: over image')" ALT=
```

## 13.10.13 Discussion of Activity 4

Your code might look something like:

```
<IMG SRC="left.gif" ALT="Left image" onMouseOver='window.status="This image"'>
```

## 13.10.14 Discussion of Activity 5

Your code might look something like:

```
<IMG SRC="left.gif" onMouseOver="this.src='right.gif' "  
onMouseOut="this.src=
```

## 13.10.15 Discussion of Activity 6

The code is changed merely by adding a variable to remember that the originally labelled button was clicked and an *if* statement to test for this and avoid doing anything if the button has previously been clicked.

```
<FORM>  
<SCRIPT>var stillToConfirm = true</SCRIPT>  
<INPUT type=button value="Click to order"  
  onClick="if (stillToConfirm){window.alert('Purchase confirmed.  
  Thank you'); this.value = '[purchase confirmed]';stillToConfirm  
  = false}">  
</FORM>
```

## 13.10.16 Answer to Review Question 1

An event is something whose occurrence you can capture by providing JavaScript code to handle it in the object where the event takes place. When you capture it, you can execute JavaScript to perform some action.

## 13.10.17 Answer to Review Question 2

This statement is true. The event occurs whether there is a handler or not. If there is a handler the code is executed. A separate event occurs when the user releases the mouse button.

## 13.10.18 Answer to Review Question 3

No, most events are not captured by JavaScript code. The browser will handle events in a default manner if no code is written. Indeed most events are simply ignored by the browser.

## 13.10.19 Answer to Review Question 4

You return true when you do not want the browser to show the anchor's link URL in the window's status area.

## 13.10.20 Answer to Review Question 5

The keyword is *this*.

## 13.10.21 Answer to Review Question 6

You have to arrange that the graphics representing the normal button and the selected button replace each other as the mouse pointer is moved over the button (or anchor) and then away again. This means you have to update the *src* property of the image object to be assigned one graphic or another.

## **13.10.22 Answer to Review Question 7**

The browser object is referred to by the variable *navigator*.

## **13.10.23 Answer to Review Question 8**

This statement is false. There are many HTML tags that can have event handlers for the `mouseover` event. `<p>`, `<div>`, and `<h1>` and a few other marks.

## **13.10.24 Answer to Review Question 9**

`setTimeout` is not an event. It is merely a function which will cause an event to happen. Other functions and actions can cause events to happen. For instance, if a function loads a new document into the current window, this will cause an `onLoad` event.

## **13.10.25 Answer to Review Question 10**

This statement is true. An HTML mark such as `<a>` represents an object. In fact, there is a very close relation between HTML marks and JavaScript objects.

# Chapter 16. JavaScript 3: Functions

## Table of Contents

Objectives.....	2
14.1 Introduction.....	2
14.1.1 Introduction to JavaScript Functions .....	2
14.1.2 Uses of Functions .....	2
14.2 Using Functions .....	2
14.2.1 Using built-in functions .....	3
14.2.2 Using user-defined functions .....	3
14.2.3 Defining and invoking a function in the same file.....	3
14.2.4 Invoking a file defined in a different file .....	3
14.2.5 Executing code using 'eval'.....	4
14.3 Creating user-defined functions.....	4
14.3.1 A simple function to display the String “hello”.....	4
14.3.2 Creating a function using function statements.....	5
14.3.3 Creating a function using the 'Function()' constructor .....	5
14.3.4 Creating a function using function literals.....	6
14.4 Some simple functions.....	6
14.4.1 Mathematical functions.....	6
14.4.2 Functions that RETURN a value .....	7
14.4.3 Defining a function that returns a value.....	8
14.4.4 A date Function.....	8
14.4.5 The today function described.....	9
14.5 Mathematical Functions.....	11
14.5.1 A form for calculations .....	11
14.5.2 A familiar calculator interface .....	13
14.5.3 Some Function activities.....	15
14.6 Form Validation.....	17
14.6.1 Testing for empty fields.....	17
14.6.2 The HTML defining the table .....	17
14.6.3 The JavaScript function to validate the form fields .....	18
14.6.4 Simplifying the code with a new function .....	19
14.6.5 Improving user interaction.....	20
14.6.6 Validation of multiple fields .....	20
14.7 Testing for numeric fields.....	21
14.8 Testing for invalid field combination.....	23
14.9 The remaining activities.....	27
14.9.1 Activity 6: Completing the colour model form.....	27
14.9.2 Activity 7: Avoiding Multiple Messages .....	27
14.10 Review Questions.....	28
14.10.1 Review Question 1 .....	28
14.10.2 Review Question 2 .....	28
14.10.3 Review Question 3 .....	28
14.10.4 Review Question 4 .....	29
14.11 Discussion Topic.....	29
14.12 Extension: More complex functions.....	29
14.13 Discussions and Answers.....	31
14.13.1 Discussion of Exercise 1 .....	31
14.13.2 Discussion of Activity 1 .....	31
14.13.3 Discussion of Activity 2.....	32
14.13.4 Discussion of Activity 3.....	32
14.13.5 Discussion of Activity 4.....	32
14.13.6 Discussion of Activity 5.....	33
14.13.7 Discussion of Activity 6.....	33
14.13.8 Discussion of Activity 7.....	33

14.13.9	Discussion of Review Question 1.....	35
14.13.10	Discussion of Review Question 2.....	35
14.13.11	Discussion of Review Question 3.....	36
14.13.12	Discussion of Review Question 4.....	36
14.13.13	Thoughts on Discussion Topic .....	36

## Objectives

At the end of this chapter you will be able to:

- Understand the importance of functions;
- Write HTML files using JavaScript functions;

## 14.1 Introduction

### 14.1.1 Introduction to JavaScript Functions

JavaScript functions are usually given a name, but since JavaScript functions are just objects in their own right, they can be stored in variables and object properties (see later unit). Functions are different from other objects in that they can be invoked (executed) with the use of a special operator (`()`).

JavaScript provides many pre-written (*built-it*) functions, for example the function *write*, provided by the *document* object (see in a later unit how related functions can be stored inside objects; as we noted a few units ago, such functions are called methods).

An example of the *write* function being invoked is as follows:

```
document.write( "This message appears in the HTML document" );
```

An example of the *alert* function being invoked is as follows:

```
alert( "This message appears in an alert dialog" );
```

Some functions return a value. For example, mathematical functions perform calculations on the provided data and return numerical results. Other functions return true/false values, or text. Some functions return no value at all, but rather perform a side-effect; *write* is one such function whose side-effect is to send text to the HTML document. Functions frequently both perform a side-effect and return a value.

### 14.1.2 Uses of Functions

Functions can be used for many different purposes. In the past, JavaScript was used to create scrolling status bar messages for sites that want to give out important information that the user may miss while browsing the site (these proved to be largely irritating to the user, and have been very rarely used in the last few years). Displaying the date and time is also a common use of JavaScript. Image mouseovers, implemented using the HTML event system described in the previous chapter, are also widely used.

Functions are very useful when used in conjunction with HTML forms. Form entry can be validated, and the input of early parts of the forms might be used to invoke functions to automatically enter appropriate data in other parts of the forms.

#### To Do

Read up about JavaScripts Functions in your textbook.

## 14.2 Using Functions

## 14.2.1 Using built-in functions

The following lines illustrate the use of built-in functions:

```
document.write( "Hello" ); document.write(
Math.sqrt( 2 ) );
document.write( "The bigger of 4 and 5 is : " + Math.bigger(4, 5) );
```

## 14.2.2 Using user-defined functions

You can define your own functions in the same file that they are invoked in, or in a different file which you can then load in a browser whenever you wish to use the function. Each of these situations are illustrated below.

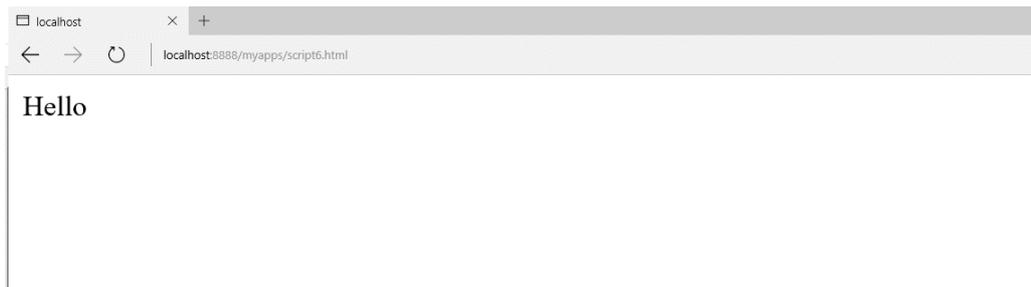
## 14.2.3 Defining and invoking a function in the same file

The following code defines and invokes a function named *displayHello*:

```
<HTML>
<SCRIPT>
////////////////////////////////////
/// define function here ///
//////////////////////////////////// function
displayHello()
{
document.write( "Hello" )
}
////////////////////////////////////
/// invoke function here ///
//////////////////////////////////// displayHello();
</SCRIPT>

</HTML>
```

The browser output when this HTML file is loaded is as follows:



## 14.2.4 Invoking a file defined in a different file

Some functions prove very useful; in order to use them in multiple Web pages they can be stored in a separate file. In the example below, the function *displayHello* has been defined in the file *helloFunction.js*. The HTML below uses two *<SCRIPT>* tags, one to load the function definition from *helloFunction.js*, and the second to invoke the function:

```
<SCRIPT SRC="helloFunction.js"></SCRIPT>
<SCRIPT> <!--
/// invoke function here /// displayHello();
</SCRIPT> -->
```

The contents of the file `helloFunction.js` is simply the JavaScript definition of the function:

```

/// define function here /// function displayHello()
{
document.write( "Hello" )
}

```

Notice that `helloFunction.js` is not an HTML file and does not contain any HTML tags. This is signified by choosing an appropriate file extension — the convention is to use the two-character extension ".js" for JavaScript files.

## 14.2.5 Executing code using 'eval'

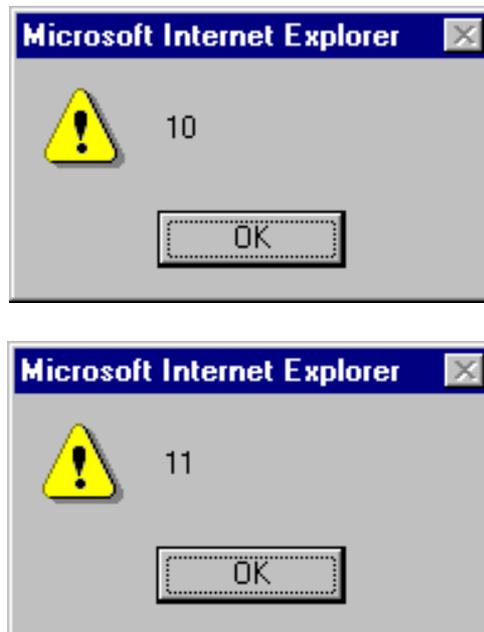
The *eval* operator expects a String containing JavaScript as an argument, and will execute the String as if it were a JavaScript statement. The code below creates a String named *myStatements* and then executes the String using *eval*:

```

var myStatements = " var n = 10; alert( n ); n++; alert( n ) " ;
eval( myStatements );

```

The result of executing this code is the two alert dialogs:



## 14.3 Creating user-defined functions

### 14.3.1 A simple function to display the String “hello”

Let's assume that we want to create a function that simply executes the following line:

```
document.write( "Hello" );
```

This function does not need any arguments, since it will do the same thing every time. The body of our function will be the single JavaScript statement above.

The table below illustrates a design for our function:

Name (optional)	
Arguments (optional)	

body	document.write("Hello")
Returns (optional)	

Depending on how we create the function, we may or may not need to name the function (in most cases it is useful to name functions).

## 14.3.2 Creating a function using function statements

The most convenient way to define a function is to use the *function* operator. The following example defines the *displayHello* function previously used:

```
function displayHello()
{
  document.write( "Hello" );
}
```

The function operator requires the following:

```
function displayHello()           -- Function name followed by list
of arguments                      -- Open Brace
{                                  -- Sequence of JavaScript statements
  document.write( "Hello" );      -- forming the
}                                  -- Close Brace
```

As can be seen above, if there are no arguments an empty pair of parentheses () is written after the function name.

Our function design looks like the following:

Name (optional)	displayHello
Arguments (optional)	
body	document.write("Hello")
Returns (optional)	

## 14.3.3 Creating a function using the 'Function()' constructor

Another way to define a function is by using the *Function()* constructor. Recall that functions are themselves just objects, and that objects are created using constructor functions. *Function()* allows a function to be defined by passing a series of *Strings* to it. All but the last *String* lists the arguments to the function, while the last *String* defines the sequence of statements that form the function's body.

The *Function()* constructor returns a function which can be assigned to a variable of your choice. So, we can now define *displayHello()* in this alternate way:

```
var displayHello = new Function( "document.write( 'Hello' );" );
```

Notice how there is no need for braces { } to be used, since the body statements are contained in a *String*. Defining functions with *Function()* constructor is less convenient than using the function operator, but is extremely useful when dynamically creating functions, since the function arguments and body can easily be created as *Strings*.

Notice the single quotes around *'Hello'* — a double quoted *String* cannot appear inside a double quoted *String*. Having single quotes on the outside, and a double quoted *"Hello"* works just as well:

```
var displayHello = new Function( 'document.write( "Hello" );' );
```

In our above call to `Function()` there is only one *String*, since `displayHello()` requires no arguments. As before, our function design looks like this:

Name (optional)	displayHello
Arguments (optional)	
body	document.write("Hello")
Returns (optional)	

### 14.3.4 Creating a function using function literals

Another third way to define a function is by using a function literal. This approach is very similar to using the function operator, except that the function is now nameless, although it can still be assigned to an arbitrary variable.

To define *displayHello* using a function literal, we would write the following:

```
var displayHello = function() { document.write( "Hello" ); }
```

Notice that function literals use a very similar syntax to function statements, and differ only in that the name of the function is not given.

#### Note

Although there are three different ways to define a function, in most situations you will find that using named functions defined with the function operator (the first technique described above) is easiest, although the other techniques all have their uses.

All the examples in the rest of this unit (and in most of the other units) define functions using the function operator.

## 14.4 Some simple functions

### 14.4.1 Mathematical functions

A JavaScript function to add two numbers:

```
function add()
{
  document.write( 5+5 );
}
add();
```

Name (optional)	add
Arguments (optional)	
body	document.write(5+5)
Returns (optional)	

And functions to perform various other arithmetical operations:

```
function minus()
{
    document.write("<p>" + (6-4) );
}
function times()
{
    document.write("<p>" + 6*4 );
}
add();
minus();
times();
```

### Note: Function naming convention

You should always name your functions and variables with a lower case first letter, unless you are writing a function that will act as a constructor (see later unit). If your function name is made up of a number of words, start the second and subsequent words with an upper case letter to make the names more readable. Examples might include:

- function calculateTaxTotal()
- function changeImage()
- function setCircleRadius()
- function divide()

## 14.4.2 Functions that RETURN a value

Most mathematical functions do not display their results in an HTML document: they only calculate and return intermediate results for use by other functions.

For example, the code below uses the *Math.pow( value, power )* function to calculate the area of a circle. Also the *Math.round()* function is used to create *roundedArea*.

```
// calculate circle ara
var radius = 10;
var circleArea = 3.1415 * Math.pow( radius, 2 );
// round to 2 decimal places
var roundedArea = Math.round( circleArea * 100 ) / 100;
document.write( "<p> Area of circle with radius " + radius + "
has area of " + circleArea );
document.write( "<p> rounded area is " + roundedArea );
```



Area of circle with radius 10 has area of 314.15000000000003

rounded area is 314.15

### 14.4.3 Defining a function that returns a value

Creating a user-defined function that returns a value is straightforward: at some point in the function's execution a return statement needs to be executed. Typically, functions returning a value require arguments that will be used in the calculation of that value.

For example we might wish to define a function *addVAT( total )* which adds 14% to a total, and returns this new value. The specification for our function might appear as follows:

Name (optional)	addVAT
Arguments (optional)	total
body	return (total * 1.14)
Returns (optional)	Value representing 14% VAT added to total

```
function addVAT( total )
{
return (total * 1.175);
}
var goodsTotal -=50;
writeln(" total before tax = " + goodsTotal );
var newTotal = addVAT(goodsTotal);
writeln("<p> total with tax added " + newTotal );
```

As can be seen, to make a function return a value we include a return statement after having done the appropriate calculation.

Arguments are named like variables (in fact they can be thought of as a kind of local variable). So these named arguments can be referred to in the processing and calculation of the function.

### 14.4.4 A date Function

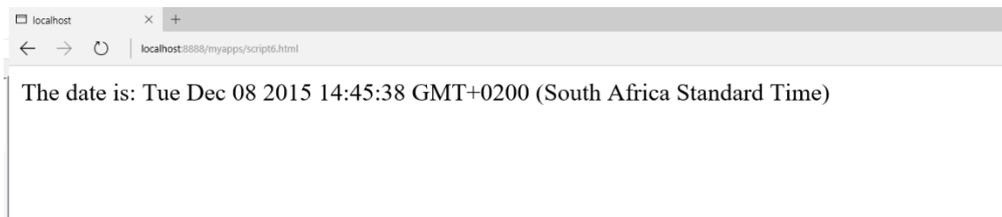
The function below displays the current date and time:

```
//function name
function today()
//begin function statements
{
//declare variable dayAndTime
// and initialise to become a new Date object
var dayAndTime = new Date()
//write date to browser page
document.write("The date is: " + dayAndTime)
//end function statements
}
//invoke function
today()
```

The above code includes many comments explaining how it functions. Without these comments the function and its invocation would look as follows:

```
function today()
{
//declare variable dayAndTime
// and initialise to become a new Date object
var dayAndTime = new Date()
//write date to browser page
document.write("The date is: " + dayAndTime)
}
today()
```

The browser output is as follows:



### Note

The behaviour of the scripts may vary according to which Web browser you use.

## 14.4.5 The today function described

This function has been defined with:

```
function today()
{
...
}
```

The first statement declares a variable called *dayAndTime* initialised with a *Date* object. *Date* was discussed in previous units, and allows access to the current date and time.

### Note

JavaScript is case sensitive. The variable *dayAndTime* is not the same as *dayandtime*.

It is a good idea to keep all your JavaScript in one case. Lowercase letters, except for the first letter of the second and later words, is the most appropriate choice (unless working with class names).

At this point, the function now has the date and time stored, but has not yet displayed it. The second statement does this:

```
document.write("The date is: " + dayAndTime)
```

In the second statement the *document* object refers to the primary Web browser window. The method *write* allows content to be written to the browser.

The output is determined in this particular instance by:

```
("The date is: " + dayAndTime)
```

The value of variable *dayAndTime* is converted to text and concatenated (added) to the String *"The date is: "*.

This new String is sent to the write statement and displayed in the browser window.

### **Exercise 1**

Open a new notepad document and write (or copy) the script for the *today* function. Save the document as "today.html" (or something similar) and load this HTML document into your Web browser. Your browser should look similar to the one in 14.4.4.

Your first task is to familiarise yourself with this function.

1. Change all uses of the function name `today()` to an alternative name. Save and reload your changed HTML file.

2. Change `dayAndTime` to an alternative name. Save and reload your changed HTML file.
3. Add the `window.status = dayAndTime;` statement. Remember to use the new name you've given the `dayAndTime` variable name.

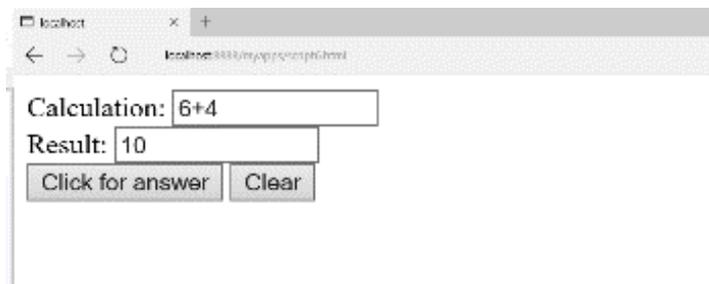
What does this new statement do? Save and reload your changed HTML file. You can find some thoughts on this exercise at the end of the unit.

## 14.5 Mathematical Functions

### 14.5.1 A form for calculations

#### The form we shall develop

We shall progress in stages to a form providing a simple calculator as follows:



The above image shows an example of a single line text field with  $4 * 7$  in it. By clicking on the *Click for answer* button the result appears in the second (*result*) text field.

There are two important events that happen on such a form:

- The user clicking the "Click for answer" button.
- The user clicking the "Clear" button.

A function that evaluates the calculation in the first text box and places the result in the second text box. The second of these events can be more simply implemented with a form *reset* button.

The other events that occur will be the user typing in a calculation in the "Calculation:" text field. The browser will ensure that the user's input is stored in the appropriate form's text field.

#### The HTML for the form

The following HTML code displays the text, the text boxes and the two buttons:

```
<FORM>
Calculation: <INPUT TYPE=text NAME=expression SIZE=15><br>
Result: <INPUT TYPE=text NAME=answer SIZE=15><br>
<INPUT TYPE=button VALUE="Click for answer"
```

```
onClick="calculate(this.form)">
<INPUT TYPE=reset VALUE="Clear">
</FORM>
```

As you can see, when the first (Click for answer) button is clicked it will call the `calculate()` function, passing it the argument `form`. This argument refers to the form the button appears in.

The second button is a standard *reset* button called 'Clear'. When a 'reset' button is clicked, the browser clears all text boxes in form.

There are two named text fields: `expression` and `answer`.

When the *Click for answer* button is clicked the JavaScript function `calculate()` is invoked, with the argument `this.form`. This argument refers to the form in which the function is being invoked from.

## The JavaScript Function

The function is quite straightforward: it retrieves the expression the user has entered in the text field named `expression`, evaluates this expression and places the answer into the `answer` text field.

The function is passed an argument referring to the form where these buttons and fields are defined. The function specification can be written as follows:

Name (optional)	<code>calculate</code>
Arguments (optional)	<code>theForm</code>
body	<pre>var result = eval( theForm.expression.value); theForm.answer.value = result;</pre>
Returns (optional)	

We can refer to the value of a text field by:

```
[formname].[fieldname].value
```

We have named our function argument *theForm* — this argument will refer to whichever form the function has been called from.

As you can see, we are evaluating the contents of the `expression` form field as follows:

```
eval( theForm.expression.value )
```

The result of this expression is assigned to a variable called `result`:

```
var result = eval( theForm.expression.value );
```

The final statement assigns the result to the `answer` field.

```
theForm.answer.value = result;
```

## The Full HTML file

The complete HTML file, including both function definition and HTML form, is as follows:

```
<HTML> <HEAD> <SCRIPT>
function calc( theForm )
{
// evaluate the text field expression;
```

```

var result = eval( theForm.expression.value);
// put result into form field 'answer'
theForm.answer.value = result;
}
// </SCRIPT> </HEAD>
<BODY>
<FORM>
Calculation: <INPUT TYPE=text NAME=expression SIZE=15><br>
Result: <INPUT TYPE=text NAME=answer SIZE=15><br>
<INPUT TYPE=button VALUE="Click for answer"
onClick="calc(this.form)">
<INPUT TYPE=reset VALUE="Clear">
</FORM>

```

### Note on eval statements

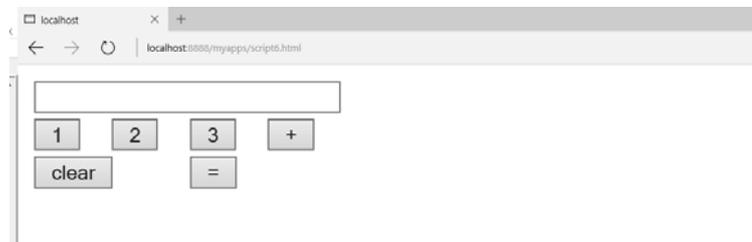
The eval statement will evaluate any JavaScript statement, not just those that perform mathematical calculations. For example, try entering `alert("Hello")` in the text box — when evaluated the browser will display the alert box.

We recommend validating the input to ensure that the expression is only mathematical before passing the entered text to eval.

## 14.5.2 A familiar calculator interface

### The HTML for the form

Let us consider a simple version of the calculator that only provides the digits 1, 2 and 3 and the addition (+) operator:



We arrange the form in a table of three rows.

The first row of the table contains a text field to display the calculation and result:

```

<TR>
<TD colspan=4><INPUT NAME=display size=30>
</TD>
</TR>

```

This first row uses a colspan of 4 to stretch over all the columns, and we name text field "display". The

second row contains four buttons (1,2,3 and +):

```

<TR>
<TD><INPUT TYPE=button VALUE=" 1 " onclick="append(this.form, 1)"
>
</TD>
<TD><INPUT TYPE=button VALUE=" 2 " onclick="append(this.form,
2) ">

```

## JavaScript 3: Functions

```
</TD>
<TD><INPUT TYPE=button VALUE=" 3 " onclick="append(this.form, 3)"
>
</TD>
<TD><INPUT TYPE=button VALUE=" + " onclick="append(this.form,
'+' )" >
</TD>
</TR>
```

Each button has an *onClick* event handler that invokes a function *append()*. *append()* is passed two arguments: *this.form* refers to the form the calculator is defined in, and the second argument is the digit (or '+' character) to be appended to the *display* text field.

The third row of the table is composed of the *clear* and = buttons:

```
<TR>
<TD colspan=2><INPUT TYPE=reset VALUE=" clear " >
</TD>
<TD colspan=2><INPUT TYPE=button VALUE=" = "
onclick="calc(this.form)" >
</TD>
</TR>
```

The *clear* button is another example of a *TYPE=reset* button. This button has been made to stretch over two columns.

The = button has an *onClick* event handler that invokes the *calc()* function. As per usual, we pass it the *this.form* form reference as an argument.

## The JavaScript Functions

Our calculator form uses two functions: the *append()* function appends a digit or symbol to the display text field, and the *calc()* function evaluates the expression in the display text field, and replaces the text in the field with the calculated answer.

The *append()* function looks like this:

Name (optional)	append
Arguments (optional)	theForm, appString
body	theForm.display.value += appStrin
Returns (optional)	

The *calc()* function is specified as follows:

Name (optional)	calc
Arguments (optional)	theForm
body	var result = eval( theForm.display.value ); theForm.display.value = result;
Returns (optional)	

The definition of the two functions follows:

```
function calc( theForm )
{
  // evaluate the text field expression;
  var result = eval( theForm.display.value );
```

### JavaScript 3: Functions

```
// put result into form field 'answer'  
theForm.display.value = result;  
}  
  
function append( theForm, appendString )  
{  
theForm.display.value += appendString  
}
```

## 14.5.3 Some Function activities

### Activity 1: Function to double a number

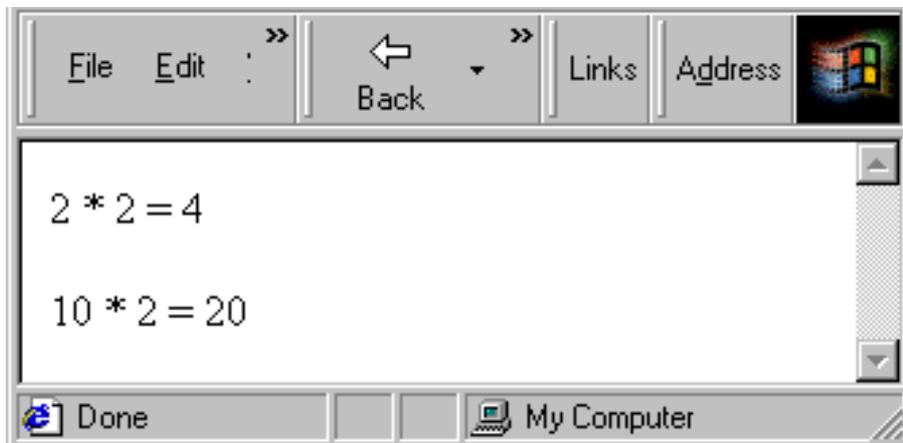
Create the specified function:

Name (optional)	displayDouble
Arguments (optional)	num
body	<pre>var result = num * 2; document.write(num + " result ");</pre>
Returns (optional)	

Invoke this function twice, first for the number 2, and a second time for the number 10.

Write a paragraph tag (<p>) to separate the displays in the browser window.

After invoking the function, the browser output should appear as follows:



You can find a discussion of this activity at the end of the unit.

### Activity 2: Function to return four times a number

Create a function to return its argument multiplied by four. This function

should be invoked as follows:

```
document.write( "<p> 4 * 2 = " + fourTimes(2) );  
document.write( "<p> 4 * 10 = " + fourTimes(10) );
```

Browser output should be the following when your function is invoked:



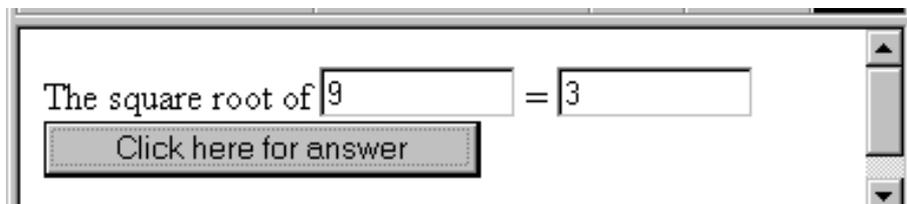
You can find a discussion of this activity at the end of the unit.

### Activity 3: Decimal places function

Define a function to return its argument rounded to two decimal places. You can find a discussion of this activity at the end of the unit.

### Activity 4: Square root calculator form

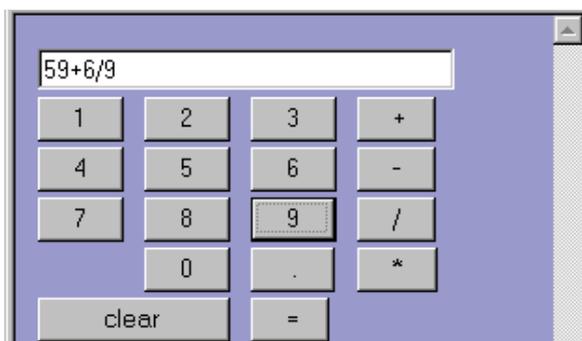
Create a form that calculates square roots. The browser should look as follows:



Hint: to calculate the square root of a number you can use *Math.sqrt()*. You can find a discussion of this activity at the end of the unit.

### Activity 5: Completing the calculator form

Extend the file smallCalculator.html so that it is a complete calculator.



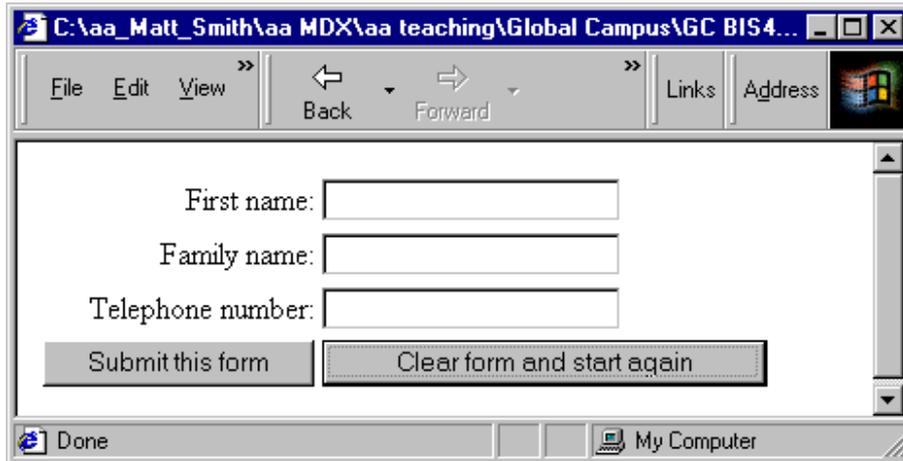
You can find a discussion of this activity at the end of the unit.

## 14.6 Form Validation

### 14.6.1 Testing for empty fields

An empty form field contains an empty String (i.e., "") as its value. Apart from seeing if the field's value is equal to the empty String, you can also test for an empty field by examining the length of the field's value. Since an empty field will have the empty String as its value, the length of the field's value will be zero.

Consider a page presenting the following form:



Let us assume that this form must have the first name and family name fields completed to be valid. The HTML

for the form is defined in a table. The form is named "order", and has an action to **post** the input values — replace *your@email.address.here* with your own email address if you wish to try out this form yourself.

```
<FORM NAME=orderform METHOD="post" ACTION="mailto:your@email.address.here"
```

The form has been defined with an *onSubmit* event handler. This handler needs to return a Boolean (true/false) value, which, if false, will prevent the form from being submitted. Therefore we need to define a function called *validateOrderForm* that returns a Boolean value *true* if the form is correct, and *false* if it is invalid in some way.

### 14.6.2 The HTML defining the table

The first row of the table displays First Name and a text input field:

```
<tr>
  <td align=right> First name: </td>
  <td>
    <INPUT TYPE="text" NAME="firstName" SIZE=20>
  </td>
</tr>
```

The text *First Name* has been right aligned to improve layout and readability of the form. The input field has been named *firstName* — *NAME="firstName"*

The second row of the table displays *Family Name* and a text input field named *familyName*:

```
<tr>
<td align=right> Family name: </td>
<td>
<INPUT TYPE="text" NAME="familyName" SIZE=20>
</td>
</tr>
```

The third row of the table displays *Telephone number* and a text input field named *telephone*:

```
<tr>
<td align=right> Telephone number: </td>
<td>
<INPUT TYPE="text" NAME="telephone" SIZE=20>
</td>
</tr>
```

The fourth row of the table displays the two buttons:

```
<tr>
<td>
<INPUT TYPE="submit" VALUE="Submit this form">
</td>
<td>
<INPUT TYPE="reset" VALUE="Clear form and start again">
</td>
</tr>
```

### 14.6.3 The JavaScript function to validate the form fields

We can test the above HTML using a dummy function that always returns false (so the form is never posted). Such a function could be written as:

```
function validateOrderForm()
{
alert( "would validate form at this point" );
// return Boolean valid data status return
false;
}
```

When we click the submit button we now see the alert dialogue appear:



To test if the *firstName* field is empty we can either compare the value of this field with an empty String:

```
orderform.firstName.value == ""
```

or we can test if the length of the value of this field is zero:

```
orderform.firstName.value.length == 0;
```

So if we wish our *validateOrderForm()* function to return true if the first name field has some value, and false otherwise we could write the function as follows:

```
function validate()
{
// at this point no invalid fields have been encountered var
fieldsValid = true;
// test field firstname
var firstNameValue = orderform.firstName.value; if
(firstNameValue.length == 0)
{
fieldsValid = false;
}
// return Boolean valid fields status return
fieldsValid;
}
```

As can be seen above, first the value of the *firstName* field is retrieved and assigned to a local variable called *firstNameValue*. Next, the length of this value is tested to see if it is zero. If it is, the Boolean variable *fieldsValid* is set to false to prevent the form from being submitted by making the function return false. Otherwise the value of this Boolean variable is left alone, and the function returns true.

If the first name field has had a value entered into, you may be informed that the form is about to be posted by the browser with a message such as the following:



## 14.6.4 Simplifying the code with a new function

The test we wrote for an empty text field is very useful for a page with many fields, so we can write a simple function called *isEmpty()* to do the test:

```
function isEmpty( fieldString )
{
if fieldString.length == 0 return true;
else
return false;
}
```

We can now rewrite the *validate()* function as follows:

```
function validate()
{
// at this point no invalid fields have been encountered var
fieldsValid = true;
// test field firstname
if isEmpty( orderform.firstName.value )
{
fieldsValid = false;
}
// return Boolean valid fields status return
fieldsValid;
}
```

## 14.6.5 Improving user interaction

While this works, the form is currently not very helpful to the user: if they click on the submit button and the first name field empty, nothing happens. At the very least the form should inform the user as to why it is not being submitted. This is easily solved by adding an alert statement, as in the following revised function definition:

```
function validate()
{
// at this point no invalid fields have been encountered var
fieldsValid = true;
// test field firstName

if ( isEmpty( orderform.firstName.value ) )
{
alert( "First name must have a value - form not submitted" ); fieldsValid =
false;
}
// return Boolean valid fields status return
fieldsValid;
}
```

Now if the submit button is pressed and the *firstName* field is empty, the form is not submitted, and the user is presented with the following dialog:



## 14.6.6 Validation of multiple fields

We can now extend our validate function to test the family name field as well:

```
function validate()
{
// at this point no invalid fields have been encountered var
fieldsValid = true;
// test field firstName
```

### JavaScript 3: Functions

```
if ( isEmpty( orderform.firstName.value ) )
{
alert( "First name must have a value - form not submitted" ); fieldsValid =
false;
}
// test field familyName
if ( isEmpty( orderform.familyName.value ) )
{
alert( "Family name must have a value - form not submitted" ); fieldsValid =
false;
}
// return Boolean valid fields status return
fieldsValid;
}
```

If the user attempts to submit the form with an empty family name:



they will be presented with the following alert dialog and the form will not be posted:



## 14.7 Testing for numeric fields

Let us assume that only digits are permitted for the telephone number field (with no spaces, or dashes or parentheses for now). Some examples of valid telephone numbers are:

```
44181362500
002356487
56478303
```

Any entry that contains non-numeric values should be considered invalid; the user should be informed of this, and the form should not be submitted.

We can create a useful function *notNumeric()* that returns true if an argument passed to it is not a number. To write the function we can make use of the special value returned by JavaScript when the result of an attempt to perform a numeric

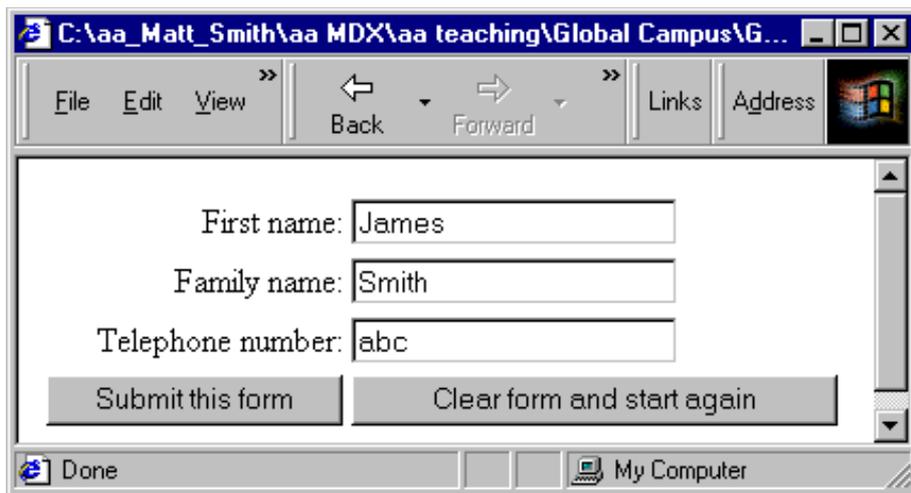
expression is not a number: JavaScript evaluates such expressions to the String "NaN". Our function can be written as follows:

```
function notNumeric( fieldString )
{
  if ( String(fieldString * 1) == "NaN" ) return true;
  else
  return false;
}
```

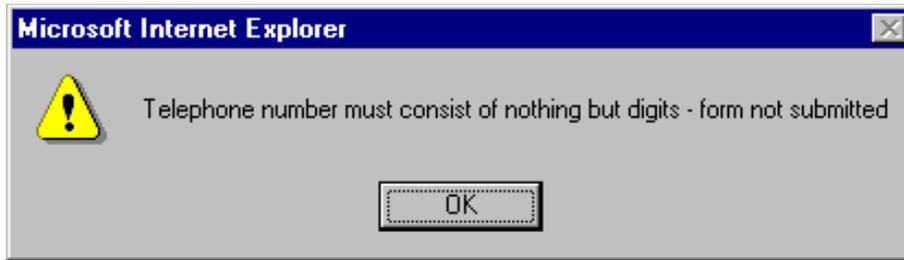
We can use this function to extend the *validate()* function to now only return true if the telephone number consists of digits:

```
function validate()
{
  // at this point no invalid fields have been encountered var
  fieldsValid = true;
  // test field firstName
  if ( isEmpty( orderform.firstName.value ) )
  {
    alert( "First name must have a value - form not submitted" ); fieldsValid
    = false;
  }
  // test field familyName
  if ( isEmpty( orderform.familyName.value ) )
  {
    alert( "Family name must have a value - form not submitted" );
    fieldsValid = false;
  }
  // test field telephone
  if ( notNumeric( orderform.telephone.value ) )
  {
    alert( "Telephone number must consist of nothing but digits - form not
    submitted" );
    fieldsValid = false;
  }
  // return Boolean valid fields status return
  fieldsValid;
}
```

So if a telephone number of "abc" is entered:

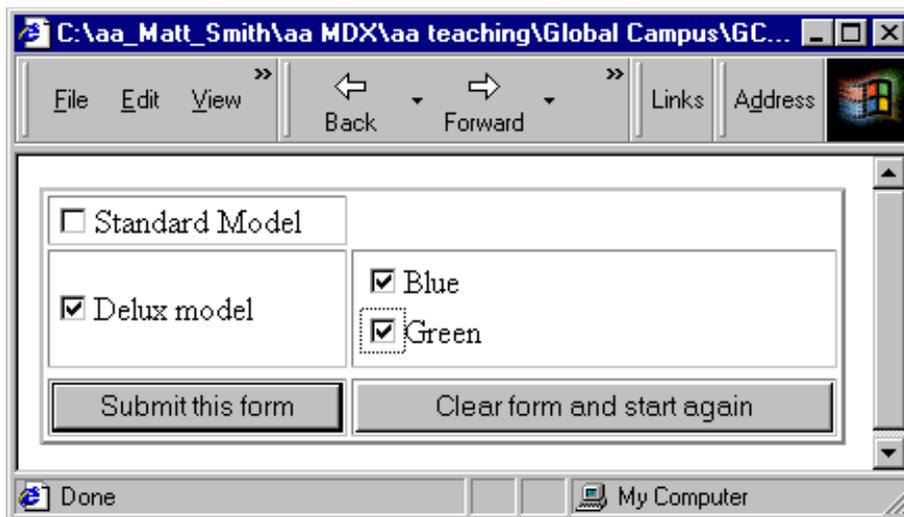


the browser will display the following alert window (and not post the form):



## 14.8 Testing for invalid field combination

Complex forms might require that only certain combinations of values/check boxes and so on be valid. The form shown below has an invalid combination of choices, and this can be picked up by a validation function. The form's submission can be cancelled and the user alerted to the problem.



On attempting to submit the form, the user is shown the following message:



The form defines three rows. The first row displays the Standard Model check box. The second row has two columns: the first contains the Deluxe model checkbox, and the second contains two colour check boxes (in a table of their own). The final row offers the submit and clear buttons as usual:

```
<FORM NAME=modelform METHOD="post"
ACTION="mailto:put.your@email.address.

<table border=2>
<tr>
<td>
<INPUT TYPE="checkbox" NAME="standard" value="Standard
model">Standard Model
</td>
</tr>
<tr>
```

### JavaScript 3: Functions

```
<td>
<INPUT TYPE="checkbox" NAME="deluxe" value="Deluxe model">Deluxe model
</td>
<td>

<table border=0>
```

```

<tr>
<td>
<INPUT TYPE="checkbox" NAME="blue">Blue
</td>
<tr>
<td>
<INPUT TYPE="checkbox" NAME="green">Green
</td>
<tr>
</table>
<tr>
<tr>
<td>
<INPUT TYPE="submit" VALUE="Submit this form">
</td>
<td>
<INPUT TYPE="reset" VALUE="Clear form and start
again">
</td>
</tr>
</table>
</FORM>

```

The *validate()* function tests for a number of invalid conditions. The first test is made to see if both the standard and deluxe models have been chosen:

```

if( modelform.standard.checked &&
modelform.deluxe.checked )
{
alert( "Please choose either standard or deluxe (not
both) - form not submitted" );
fieldsValid = false;
}

```

The next two tests examine if a colour has been chosen with the standard model (this is not permitted):

```

if( modelform.standard.checked &&
modelform.blue.checked )
{
alert( "Sorry - colour choices are only possible with deluxe
cars - form fieldsValid = false;
}
if( modelform.standard.checked &&
modelform.green.checked )
{
alert( "Sorry - colour choices are only possible with deluxe
cars - form fieldsValid = false;
}

```

The final test ensures that only a single colour has been selected:

```

if( modelform.blue.checked &&
modelform.green.checked )
{
alert( "Please either blue or green (not both) - form not
submitted" ); fieldsValid = false;
}

```

```
}
```

## 14.9 The remaining activities

### 14.9.1 Activity 6: Completing the colour model form

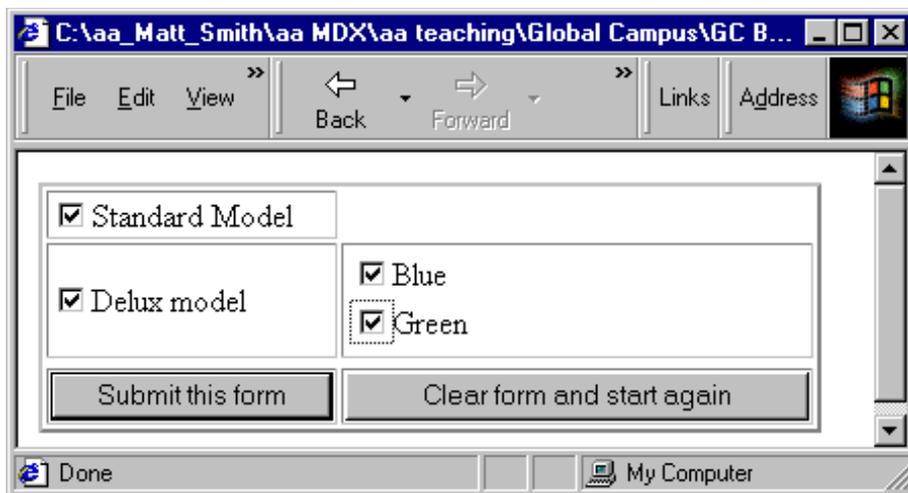
Extend the file form5.html so that it does not permit the form to be submitted when neither a standard nor a deluxe model have been selected.

You can find a discussion of this activity at the end of the unit.

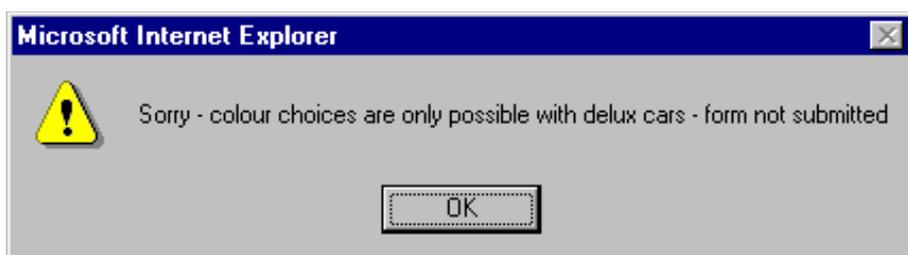
### 14.9.2 Activity 7: Avoiding Multiple Messages

It can be very annoying to users to receive many different error messages from the same form (i.e. three or four alerts all appearing one after the other).

For example, in the previous colour model form, if the screen were as follows:



the user would have to respond to the following sequence of alerts:





Amend the code so that an alert is only displayed for the first error encountered.

You can find a discussion of this activity at the end of the unit.

## 14.10 Review Questions

### 14.10.1 Review Question 1

What are the names of functions in the code below:

```
function displayMessage( message )
{
document.write( "<p> Message was: " + message );
}
displayMessage( "Hello" );
displayMessage( Math.sqrt( 25) );
```

You can find the answer to this question at the end of the unit.

### 14.10.2 Review Question 2

Write a specification for a function `triangleArea` to calculate and return the area of a triangle, for a provided height and width.

You can find the answer to this question at the end of the unit.

### 14.10.3 Review Question 3

What code would you need to create the following user-defined function:

Name (optional)	multiply
Arguments (optional)	n1, n2
body	var result = n1 * n2; return result;
Returns (optional)	Returns the result of multiplying the two given numbers

You can find the answer to this question at the end of the unit.

## 14.10.4 Review Question 4

What value is displayed when the following minus() function is invoked?

```
function minus()
{
document.write( 5 - 6 - 7 )
}
minus();
```

You can find the answer to this question at the end of the unit.

## 14.11 Discussion Topic

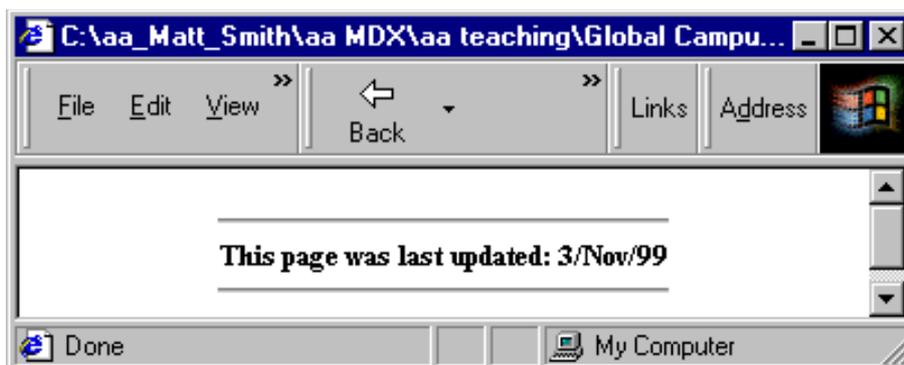
Generally, the definition of functions using the Function constructor and function literals is unnecessary since all functions can be defined using simple function statements such as the following:

```
function myFunction( arg1, arg2 )
{
// body of function
// optional function RETURN statement
}
```

You can find some thoughts on this discussion topic at the end of the unit.

## 14.12 Extension: More complex functions

A good example of a function that uses more complex JavaScript features, such as arrays and objects (see later unit), is a function to perform the standard task of displaying the date which a Web page was last updated. For example a 'banner' can be displayed at the end of each document as follows:



The code to create such output is as follows:

```

<HTML> <SCRIPT> <!--
//function called update()
function update()
{
//declare a variable called
// (Modified to equal date of last save)
var modified = document.lastModified;
var months = new
Array("Jan","Feb","Mar","Apr","May","June","July","Aug",
"Sept","Oct","Nov","Dec");
//declare a variable called ModDate to equal last modified
date
var modDate = new Date( modified );
//write string of html formatting
document.write('<center><hr width=200><font size=2><b>');
document.write('This page was last updated: ');
//write day
document.write( modDate.getDate() + '/' );
//write month
document.write( months[ modDate.getMonth() ] + '/' );
//write year
document.write(modDate.getYear());
//write string of html formatting
document.write('</b></font><br><hr width=200></center>');
}
//invoke function
update()
// --> </SCRIPT> </HTML>

```

You may wish to examine this function now, and perhaps revisit it after working through the arrays and objects unit later in this module.

## 14.13 Discussions and Answers

### 14.13.1 Discussion of Exercise 1

The browser output now has the details of whatever day and time you opened the document in the status bar.

### 14.13.2 Discussion of Activity 1

We can define this function easiest using a function statement:

```

function displayDouble( num )
{
var result = num * 2;
document.write( num + " * 2 = " + result );
}

```

The function can be invoked with statements such as:

```
displayDouble( 2 );
```

```
document.write("<p>");
displayDouble( 10 );
```

### 14.13.3 Discussion of Activity 2

We can define this function easiest using a function statement:

```
function fourTimes ( num )
{
  var result = num * 4;
  return result;
}
```

### 14.13.4 Discussion of Activity 3

The simplest solution to this is to multiple the number by 100, round this value, then divide by 100 again.

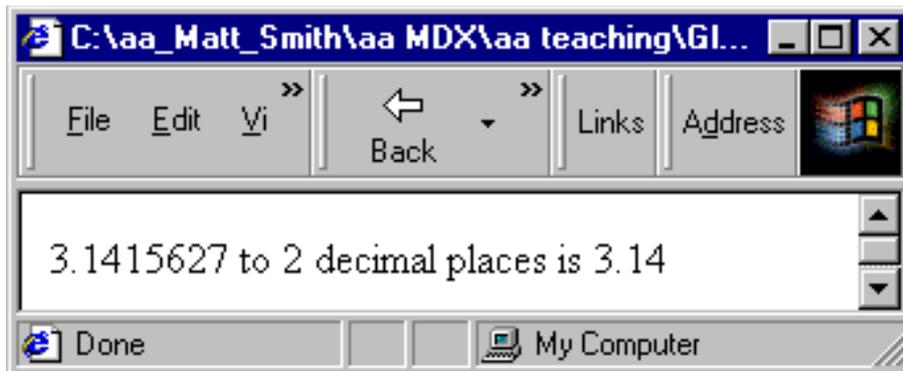
The function is as follows:

```
function twoDP( num )
{
  var rounded = Math.round( num * 100 );
  return rounded / 100;
}
```

An example of the function being invoked is as follows:

```
document.write(" 3.1415627 to 2 decimal places is " + twoDP(
3.1415627 ) );
```

The browser output should appear as follows:



### 14.13.5 Discussion of Activity 4

The code for the form should look similar to the following:

```
<form name="form2">
  The square root of
```

```

<INPUT TYPE="text" NAME="number" VALUE="" SIZE=10>
= <INPUT TYPE="text" NAME="square" VALUE="" SIZE=10><br>
<input type="button" value="Click here for answer"
onClick="display()">
</form>

```

The code for the function should look similar to the following:

```

function display( square )
{
var num = document.form2.number.value;
document.form2.square.value = Math.sqrt( num );
}

```

### 14.13.6 Discussion of Activity 5

This is straightforward — all that needs to be done is to add another three rows of buttons to the form (with appropriate `onClick` event handlers). The existing functions can be left unchanged.

The extra HTML lines are:

```

<TR>
<TD><INPUT TYPE=button VALUE=" 4 " onclick="put(this.form,
4)"></TD>
<TD><INPUT TYPE=button VALUE=" 5 " onclick="put(this.form,
5)"></TD>
<TD><INPUT TYPE=button VALUE=" 6 " onclick="put(this.form,
6)"></TD>
<TD><INPUT TYPE=button VALUE=" - " onclick="put(this.form, '-
')"></TD>
</TR>
<TR>
<TD><INPUT TYPE=button VALUE=" 7 " onclick="put(this.form,
7)"></TD>
<TD><INPUT TYPE=button VALUE=" 8 " onclick="put(this.form,
8)"></TD>
<TD><INPUT TYPE=button VALUE=" 9 " onclick="put(this.form,
9)"></TD>
<TD><INPUT TYPE=button VALUE=" / " onclick="put(this.form,
'/')"></TD>
</TR>

```

### 14.13.7 Discussion of Activity 6

We need to add a new section to the `validate()` function, testing to see if neither are selected:

```

if( !modelform.standard.checked && !modelform.deluxe.checked )
{
alert( "You must choose either standard or deluxe - form not
submitted" ); fieldsValid = false;
}

```

Note the use of the exclamation mark `!` — this is a *logical not* in JavaScript.

### 14.13.8 Discussion of Activity 7

One solution is to only display an alert for an invalid field if all previous fields have been valid. For example, we

could amend the testing of the multiple colour fields to the following:

```

if( modelform.blue.checked && modelform.green.checked )
{
if (fieldsValid)
alert( "Please either blue or green (not both) - form not submitted" );
fieldsValid = false;
}

```

The same approach needs to be taken for all but the first invalid field.

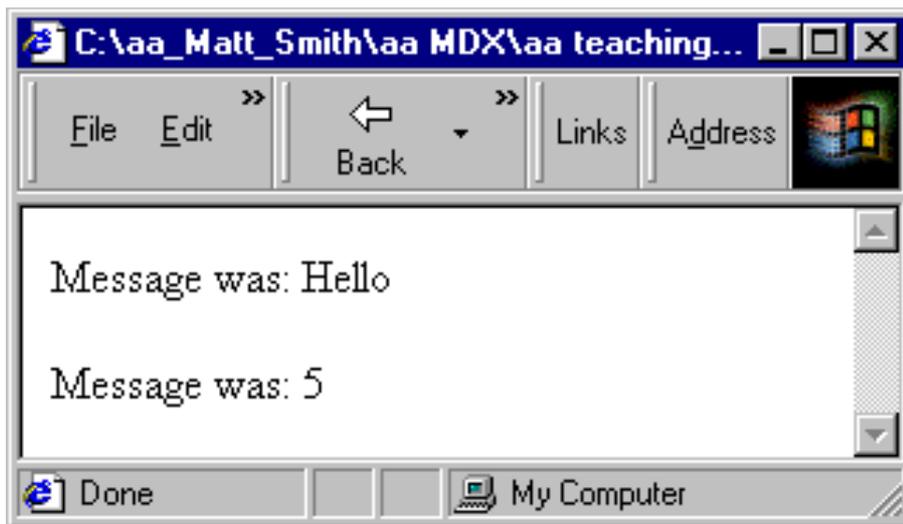
## 14.13.9 Discussion of Review Question 1

There are three functions referred to in this code:

1. The user-defined function *displayMessage()*
2. The built-in *write()* function (part of object *document* — see later unit)
3. The built-in *sqrt()* function (part of class *Math* — see later unit)

Generally, wherever you see parentheses, either function arguments are being defined, or a function is being invoked.

The browser output for the above code is:



## 14.13.10 Discussion of Review Question 2

This function should be named *triangleArea*.

*triangleArea* takes two arguments, which we shall call *height* and *width*.

The function needs to calculate the area of a triangle ( $1/2 * (width * height)$ ). We can write a statement that assigns the result of this calculation into a variable called *area*:

```

var area = 0.5 * (height * width); The
function is to return the area: return
area;

```

The specification looks as follows:

Name (optional)	triangleArea
Arguments (optional)	height, width
body	<pre>var area = 0.5 * (height * width) return area;</pre>
Returns (optional)	Returns triangle area (0.5 * (height * width) )

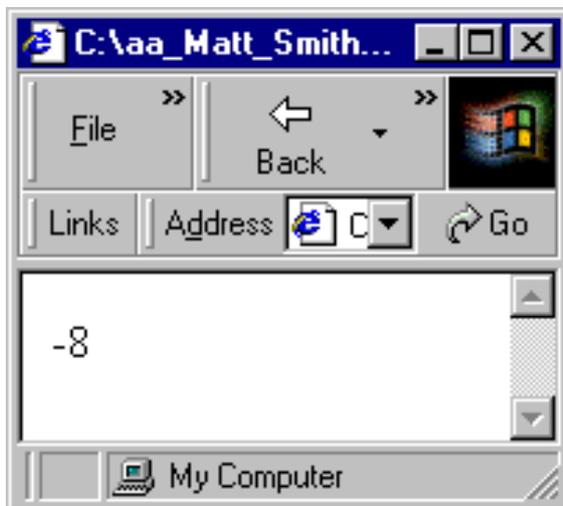
```
function triangleArea
{
var area = 0.5 (height * width) return area;
}
```

### 14.13.11 Discussion of Review Question 3

This function is most easily created using a function statement as follows:

```
function multiply( n1, n2 )
{
var result = n1 * n2; return result;
}
```

### 14.13.12 Discussion of Review Question 4



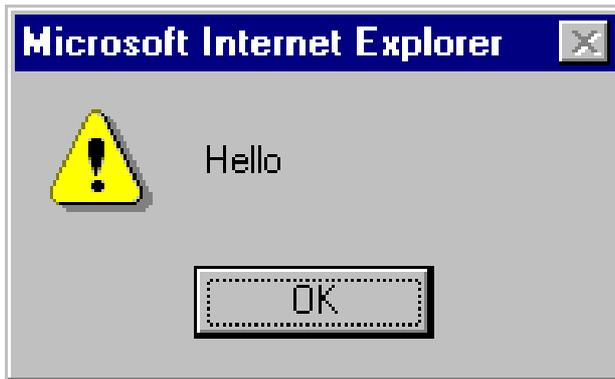
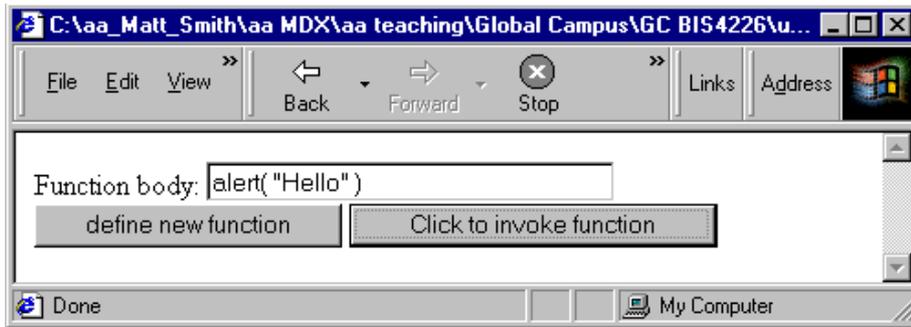
The function is invoked, the expression inside *write()* is evaluated to -8, and then the *document.write()* function is executed adding -8 to to the browser window.

### 14.13.13 Thoughts on Discussion Topic

While in most cases it is convenient to define functions using function statements, there are situations when function literals and the Function constructor offer advantages.

For example, it is possible to have collections of functions that do not require names (see later unit on arrays and objects), in which case it is frequently convenient never to have to name them.

The Function constructor offers far more power in defining functions, since it defines the arguments and body of a function using Strings, and Strings are very easy to manipulate in JavaScript. This makes it possible to have the script itself create new functions as they are needed. Consider the browser output below:



This output is possible using a String entry from the user to define a function that can then be invoked through a button press. The code for the above is as follows:

```
<HTML> <HEAD> <SCRIPT> <!--
function userFunction()
{
alert( "no function yet defined" )
}

function buildFunction( theForm )
{

// DEFINE new function and replace existing 'userFunction'
userFunction = new Function( theForm.functionbody.value );
}

// --> </SCRIPT> </HEAD>
<BODY>
<FORM>

Function body: <INPUT TYPE=text NAME=functionbody SIZE=30><br>

<INPUT TYPE=button VALUE="define new function"

onClick="buildFunction( this.form )">
<INPUT TYPE=button VALUE="Click to invoke function"
onClick="userFunction()">
</FORM>
</BODY>
</HTML>
```

Although this particular page could have been more simply implemented using `eval`, it illustrates the flexibility of the Function constructor approach to function definition — a script can be written to define functions using statements not known when the script was written.

# Chapter 15. JavaScript 4: Objects and Arrays

## Table of Contents

Objectives .....	2
15.1 Introduction.....	2
15.2 Arrays.....	2
15.2.1 Introduction to arrays.....	2
15.2.2 Indexing of array elements .....	3
15.2.3 Creating arrays and assigning values to their elements .....	3
15.2.4 Creating and initialising arrays in a single statement.....	4
15.2.5 Displaying the contents of arrays.....	4
15.2.6 Array length.....	4
15.2.7 Types of values in array elements.....	6
15.2.8 Strings are NOT arrays .....	7
15.2.9 Variables — primitive types and reference types .....	8
15.2.10 Copying primitive type variables.....	9
15.2.11 Copying reference type variables.....	9
15.2.12 Array activities.....	10
15.2.13 Arrays concept map .....	12
15.3 The JavaScript object model .....	13
15.3.1 Introduction to the JavaScript object model .....	13
15.3.2 Objects in JavaScript .....	13
15.3.3 Naming conventions .....	14
15.3.4 Creating objects and variables referring to objects.....	15
15.3.5 Object properties and methods .....	16
15.3.6 Some important JavaScript objects.....	19
15.3.7 The 'Triangle' object .....	21
15.3.8 User defined objects .....	21
15.3.9 Implementing instance variables and methods .....	23
15.3.10 The JavaScript object search chain .....	25
15.3.11 Object activities .....	28
15.4 Arrays as objects .....	29
15.4.1 Array method: join.....	29
15.4.2 Array method: sort.....	30
15.4.3 Array method: reverse .....	31
15.4.4 Single and multi-dimensional arrays .....	32
15.5 Objects as associative arrays .....	33
15.5.1 Enumerating associative arrays with FOR/IN loop statements.....	33
15.5.2 Using FOR/IN for numerically index arrays.....	34
15.5.3 Activities and further work.....	36
15.6 Review Questions .....	41
15.7 Discussion Topics .....	42
15.8 Answers.....	42
15.8.1 Discussions of Exercise 1 .....	43
15.8.2 Discussions of Exercise 2 .....	43
15.8.3 Discussions of Activity 1 .....	44
15.8.4 Discussions of Activity 2.....	44
15.8.5 Discussions of Exercise 3 .....	45
15.8.6 Discussions of Activity 3.....	45
15.8.7 Discussions of Activity 4.....	47
15.8.8 Discussions of Activity 5.....	47
15.8.9 Discussions of Activity 6.....	48
15.8.10 Discussions of Activity 7.....	48

15.8.11	Discussions of Activity 8.....	49
15.8.12	Answers to Review Questions .....	51
15.8.13	Contribution to Discussion Topics.....	51

# Objectives

At the end of this chapter you will be able to:

- Understand the basic features of JavaScript arrays;
- Understand the fundamental elements of JavaScript arrays;
- Write HTML files using JavaScript arrays;
- Explain the JavaScript object model;
- Use arrays as objects.

## 15.1 Introduction

Most high level computer programming languages provide ways for groups of related data to be collected together and referred to by a single name. JavaScript offers objects and arrays for doing so. JavaScript arrays are rather different from arrays in many programming languages: all arrays are objects (as in many other languages), but they are also *associative* arrays. Also, all objects can also be used as if they, too, are arrays.

This chapter is organised into four sections. It introduces arrays and objects separately, then considers arrays as objects, then finally considers objects as (associative) arrays. Many important concepts are covered in this unit, although much of the object technology concepts have been introduced in earlier units.

When working with variables, an important distinction has to be made: does the variable contain the value of a primitive type, or does it contain a reference to a (non-primitive) collection of data. A thorough grounding in the concepts covered in this chapter is necessary to both be able to understand the sophisticated JavaScript scripts written to support complex websites, and to be able to begin developing JavaScript solutions yourself for real world problems. It is important to work through examples until you understand them; write your own programmes that use and test your learning. Programming is learnt through doing as much, or more so, than by reading.

## 15.2 Arrays

### 15.2.1 Introduction to arrays

Arrays provide a tabular way to organise a collection of related data. For example, if we wished to store the seven names of each weekday as Strings, we could use an array containing seven elements. This array would be structured as follows:

Index	Value
weekDays[0]	"Monday"
weekDays[1]	"Tuesday"
weekDays[2]	"Wednesday"
weekDays[3]	"Thursday"
weekDays[4]	"Friday"
weekDays[5]	"Saturday"
weekDays[6]	"Sunday"

As can be seen all of these different String values are stored under the collective name `weekDays`, and a number (from 0 to 6) is used to state which of these `weekDays` values we specifically wish to refer to. So by referring to `weekDays[3]` we could retrieve the String "Thursday".

#### DEFINITION - Array

An array is a tabular arrangement of values. Values can be retrieved by referring to the array name together with the numeric index of the part of the table storing the desired value.

As you may have spotted, by having a loop with a numeric variable we can easily perform an action on all, or some sub-sequence, of the values stored in the array.

## 15.2.2 Indexing of array elements

As can be seen from the above figure, there are seven elements in the `weekDays` array.

### DEFINITION — element

Arrays are composed of a numbered sequence of elements. Each element of an array can be thought of as a row (or sometimes column) in a table of values.

The seven elements are indexed (numbered) from zero (0) to six (6). Although it might seem strange to start by numbering the first element at zero, this way of indexing array elements is common to many high-level programming languages (include C, C++ and Java), and has some computational advantages over arrays that start at 1.

### Note

The index of an array element is also known as its subscript. The terms array index and array subscript can be used interchangeably. In this unit we consistently use the term *index* for simplicity.

### Exercise 1

Answer the following questions about the `weekDays` array:

- What is the first element?
- What is the last element?
- What is the 4th element?
- What is the value of the first element?
- What is the value of the 4th element?
- What is the element containing String "Monday"?
- What is the element containing String "Saturday"?
- What is the index of the element containing String "Monday"?
- What is the index of the element containing String "Saturday"?

## 15.2.3 Creating arrays and assigning values to their elements

There are a number of different ways to create an array. One piece of JavaScript code that creates such an array is as follows:

```
// VERSION 1
var weekDays = new Array(7); weekDays[0] = "Monday"; weekDays[1] =
"Tuesday"; weekDays[2] = "Wednesday"; weekDays[3] = "Thursday";
weekDays[4] = "Friday"; weekDays[5] = "Saturday"; weekDays[6] =
"Sunday";
```

The first (non-comment) line is:

```
var weekDays = new Array(7);
```

This line declares a new variable called `weekDays` and makes this new variable refer to a new `Array` object that can hold seven elements.

## Note

The concept of arrays as objects is discussed later this unit.

The seven statements that follow this line assign the Strings `"Monday"` - `"Sunday"` to the array elements `weekDays[0]` to `weekDays[6]` respectively.

## 15.2.4 Creating and initialising arrays in a single statement

Another piece of JavaScript that would result in the same array as VERSION 1 above is the following:

```
// VERSION 2 - all in one line
var weekDays = new Array( "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday", "Sunday" );
```

This single statement combines the declaration of a new variable and `Array` object with assigning the seven weekday Strings. Notice that we have not had to specify the size of the array, since JavaScript knows there are seven Strings and so makes the array have a size of seven elements.

The above examples illustrate that arrays can either be created separately (as in VERSION 1), and then have values assigned to elements, or that arrays can be created and provided with initial values all in one statement (VERSION 2).

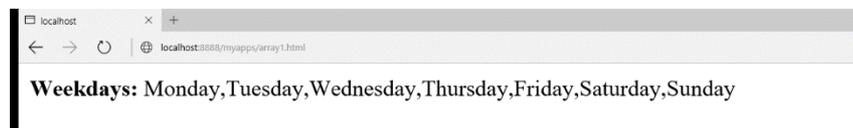
When declaring the array, if you know which values the array should hold you would likely choose to create the array and provide the initial values in one statement. Otherwise the two-stage approach of first creating the array, and then later assigning the values, is appropriate.

## 15.2.5 Displaying the contents of arrays

The easiest way to display the contents of an array is to simply use the `document.write()` function. This function, when given an array name as an argument, will display each element of the array on the same line, separated by commas. For example, the code:

```
var weekDays = new Array( "Monday", "Tuesday", "Wednesday", "Thursday",
    "Friday", "Saturday", "Sunday" );
document.write( "<b> Weekdays: </b>" + weekDays );
```

Produces the following output in a browser:



## 15.2.6 Array length

The term *length*, rather than *size*, is used to refer to the number of elements in array. The reason for this will become clear shortly.

As illustrated in the VERSION 1 code above, the size of an array can be specified when an array is declared:

```
var weekDays = new Array(7);
```

This creates an array with seven elements (each containing an undefined value):

Index	Value
weekDays[0]	<undefined>
weekDays[1]	<undefined>
weekDays[2]	<undefined>
weekDays[3]	<undefined>
weekDays[4]	<undefined>
weekDays[5]	<undefined>
weekDays[6]	<undefined>

In fact, while this is good programming practice, it is not a requirement of the JavaScript language. The line written without the array size is just as acceptable to JavaScript:

```
var weekDays = new Array();
```

this creates an array, with no elements:

Index	Value

In this second case, JavaScript will make appropriate changes to its memory organisation later, once it identifies how many elements the array needs to hold. Even then, JavaScript is a can extend the size of an array to contain more elements than it was originally defined to contain.

For example, if we next have the statement:

```
weekDays[4] = "Friday";
```

the JavaScript interpreter will identify the need for the weekDays array to have at least five elements, and for which the 5th element is the String "Friday".

Index	Value
weekDays[0]	<undefined>
weekDays[1] weekDays[2]	<undefined> <undefined>
weekDays[3]	<undefined>
weekDays[4]	"Friday"

If this were then followed by the statement:

```
weekDays[6] = "Sunday";
```

the JavaScript interpreter will identify the need for the weekDays array to now have seven elements, of which the 5th contains the String "Friday" and the 7th contains "Sunday":

Index	Value
weekDays[0]	<undefined>
weekDays[1]	<undefined>
weekDays[2]	<undefined>
weekDays[3]	<undefined>
weekDays[4]	"Friday"

Index	Value
weekDays[5]	<undefined>
weekDays[6]	"Sunday"

Once created, an array has a length property. This stores the number of elements for which JavaScript has made space. Consider the following statements:

```
var weekDays = new Array(7); var months = new Array();
var bits = new Array(17, 8, 99);
```

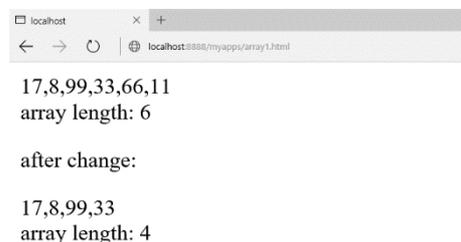
The length of each of these arrays is as follows:

- weekDays.length = 7
- months.length = 0
- bits.length = 3

One needs to be careful, though, since making the length of an array smaller can result in losing some elements irretrievably. Consider this code and the following output:

```
var bits = new Array(17, 8, 99, 33, 66, 11);
document.write(bits);
document.write("<br> array length: " + bits.length);
bits.length = 4;
document.write("<p> after change: </p>");
document.write(bits);
document.write("<br> array length: " + bits.length);
```

The browser output from such code is:



As can be seen, after the statement bits.length = 4; the last two array elements have been lost.

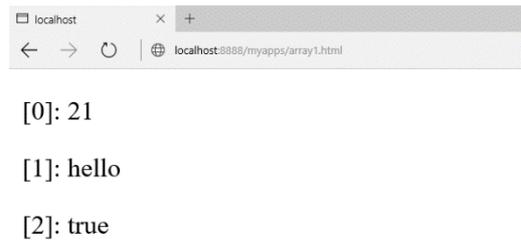
## 15.2.7 Types of values in array elements

In most high-level programming languages arrays are typed. This means that when an array is created the programmer must specify the type of the value to be stored in the array. With such languages, all elements of an array store values of a single type. However, JavaScript is not a strongly typed programming language, and a feature of JavaScript arrays is that a single array can store values of different types.

Consider the following code:

```
var things = new Array(); things[0] = 21;
things[1] = "hello";
things[2] = true;
document.write("<p>[0]: " + things[0]);
document.write("<p>[1]: " + things[1]);
document.write("<p>[2]: " + things[2]);
```

As can be seen, this is perfectly acceptable JavaScript programming:



When the value of an array element is replaced with a different value, there is no requirement for the replacement value to be of the same type. In the example below, array element 1 begins with the String "hello", is then changed to the Boolean value false, and changed again to the number 3.1415:

```
var things = new Array(); things[0] = 21;  
things[1] = "hello"; things[2] = true; things[1] = false; things[1]  
= 3.1415;  
document.write("<p>[0]: " + things[0]);  
document.write("<p>[1]: " + things[1]);  
document.write("<p>[2]: " + things[2]);
```

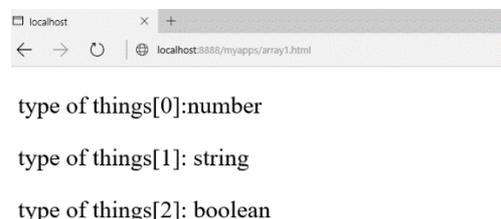
As can be seen, this changing of array element values of different types works without problems:



We can confirm that a single array can store values of different types by displaying the return value of the `typeof` function:

```
var things = new Array(); things[0] = 21;  
things[1] = "hello";  
things[2] = true;  
document.write("<p>type of things[0]:" + typeof things[0] );  
document.write("<p>type of things[1]: " + typeof things[1] );  
document.write("<p>type of things[2]: " + typeof things[2] );
```

The output of the above code is as follows:



## 15.2.8 Strings are NOT arrays

In many programming languages, text strings are represented as arrays of characters. While this makes sense in non-object oriented languages, there are a number of advantages of representing data such as text as objects (see later this unit).

You will only obtain *undefined* values if you attempt to refer to particular characters of Strings using the square bracket array indexing syntax.

For example, the code

```
var firstName = "Matthew";
document.write("second letter of name is: " + firstName[1]);
```

It is also easy to confuse String and Array objects because they both have a length property. So the code:

```
var firstName = "Matthew";
document.write("second letter of name is: " + firstName[1]);
document.write("<p> length of 'firstName' " + firstName.length);
```

is valid, and we do see the number of characters of the String displayed:

However, the similarity is because both *Strings* and *Arrays* are objects — see later in this unit for a detailed discussion of JavaScript objects.

## 15.2.9 Variables — primitive types and reference types

When one begins to work with collections of values (e.g. with arrays or objects), one needs to be aware of the variable's value. A variable containing a primitive type value is straightforward: for example consider the numeric variable age in this code:

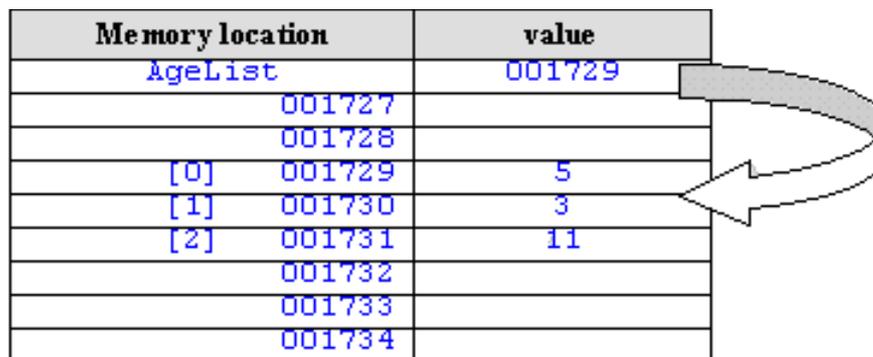
```
var age = 21;
```

Memory location	value
age	21

However, the situation is not so simple when we consider an array variable. For example the ageList array of three ages defined as follows:

```
var ageList = new Array( 3 ); ageList[0] = 5;
ageList[1] = 3;
ageList[2] = 11;
```

It is not the case that ageList is the name for a single location in memory, since we know that this array is storing three different values. The location in memory named ageList is actually a reference to the place in memory where the first value in the array can be found. The diagram below attempts to illustrate this:



### Note

There is nothing special about the locations 001727 etc., these numbers have been made up and included to illustrate unnamed locations in memory.

So we can think of the variable ageList as referring to where the array values can be found.

The implication of the different between variables of primitive types and reference types is rather important,

especially in the case of copying or overwriting the values of reference variables.

## 15.2.10 Copying primitive type variables

With primitive type variables it is straightforward to copy and change values. Consider the following code:

```
var name1 = "ibrahim"; var name2 = "james";
```

Initially the memory looks as follows:

Memory Location	Value
name1	"Ibrahim"
name2	"James"

Then if we execute the following lines:

```
name1 = name2; name2 = "fred";
```

the values in memory will be:

Memory Location	Value
name1	"James"
name2	"fred"

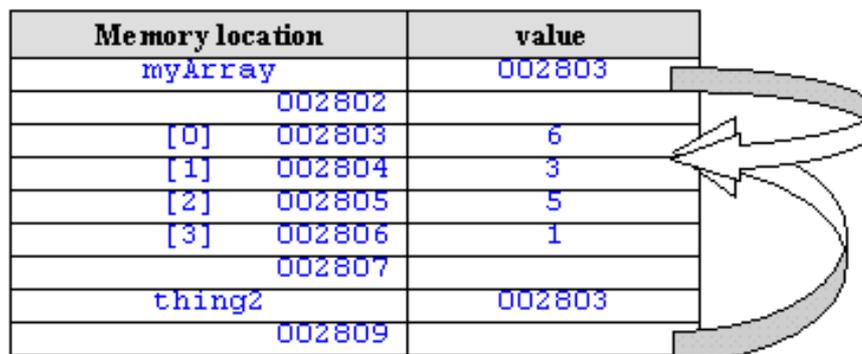
What is happening is that when one variable is assigned the value of a second, a copy of the second variable's value is placed into the location in memory for the first — so a copy of the value "james" from name2 was copied into the location for name1.

## 15.2.11 Copying reference type variables

Things are different when one is working with variables of reference types. Consider the following code, where first an array called *myArray* is created with some initial values, next a variable called *thing2* is assigned the value of *myArray*:

```
var myArray = new Array( 6, 3, 5, 1 ); var thing2 = myArray;
```

Since *myArray* is a reference to the array values in memory, what has been copied into *thing2* is the reference — so now both *myArray* and *thing2* are referring to the same set of values in memory:



The implications are that if a change is made to the array values, since both *myArray* and *thing2* are referring to the same values in memory, both will be referring to the changed array.

### Exercise 2

What is the value of `myArray[2]` after the following statements have been executed?

```
var myArray = new Array( 6, 3, 5, 1 ); var thing2 = myArray;  
  
thing2[1] = 27;  
thing2[2] = 19;  
thing2[3] = 77;
```

## 15.2.12 Array activities

### Activity 1: Creating and displaying elements of an array

Write an HTML file that creates an array called `ages` containing the following numbers: 21, 14, 33, 66, 11. Write code so that the values of array elements 0, 1 and 3 are displayed. The screen should look similar to the following when the page is loaded:



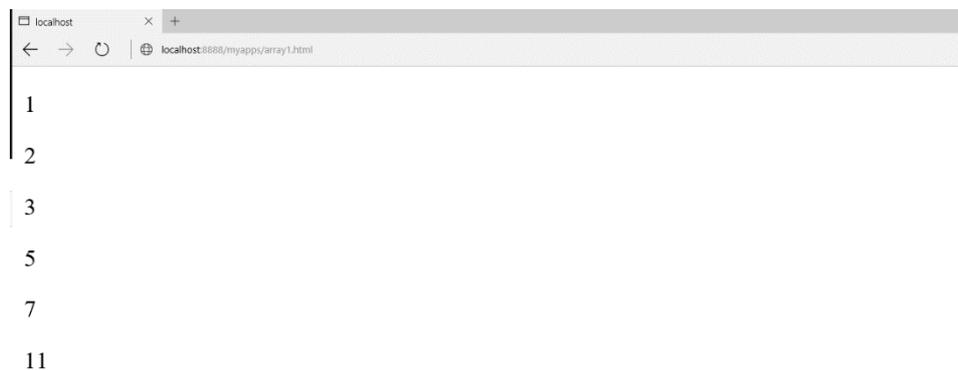
You can find a discussion of this activity at the end of the unit.

### Activity 2: Iterating through array contents

Create an array called `primes` containing the following numbers: 1, 2, 3, 5, 7, 11.

Write a while loop that will iterate (loop) through each element of the array, displaying each number on a separate line. Your code should be written in a general way, so if more numbers were added to the array (say 13, 17 and 19) the loop code would not need to be changed.

The browser output should be something like:

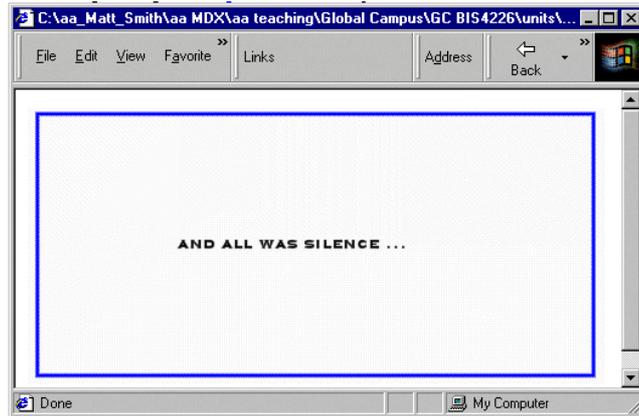


You can find a discussion of this activity at the end of the unit.

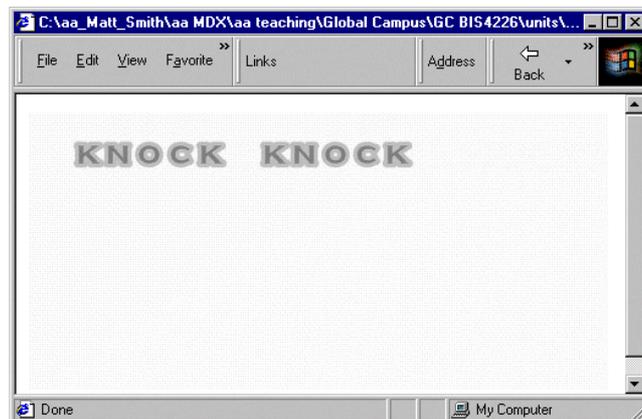
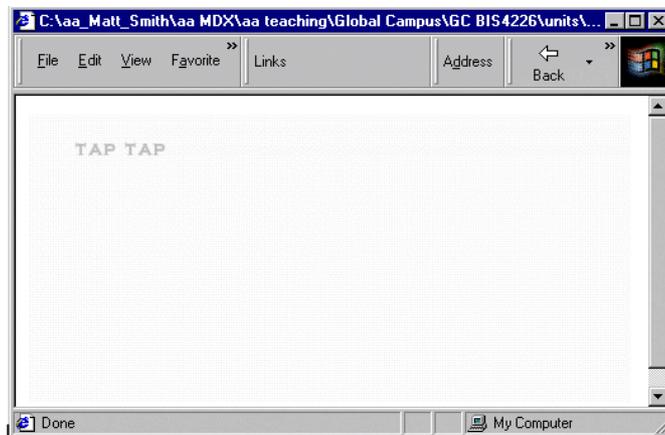
### Activity 3: Animation using an array of Image objects

Create a Web page that shows an image, and when the mouse goes over this image the image is replaced with an

animation of 3 other images before returning back to the original image. Make the original image: silence.gif, with browser output as follows:



Make the three images that are animated: taptap.gif, knockknock.gif and BANGBANG.gif, with browser outputs as follows:





Create an array of Image objects called images. You only need to refer to the src property as in the following lines:

```
// set up image array images[0] = new Image();
images[0].src = "taptap.gif";
```

In the body section of the HTML file display the silence image (naming it noise) with an appropriate *onMouseOver* JavaScript one-liner:

```
<BODY>
IMG          NAME="noise"          SRC="silence.gif"
onMouseOver="startAnimation()">
</BODY>
```

Create a function *startAnimation()* that sets the delay time and the imageNumber to display first in the animation. Your *startAnimation()* function should then make a call to an *animate()* function written as follows:

```
function animate()
{
  document.noise.src = images[ imageNumber ].src;
  imageNumber++;
  delay += 250;

  if( imageNumber < 4 )

  setTimeout( "animate()", delay );

  // if imageNumber = 4 the animation has finished
  // and this function can terminate
}
```

You might create a simple version with an unchanging delay of 500 milliseconds initially. Notice how the line:

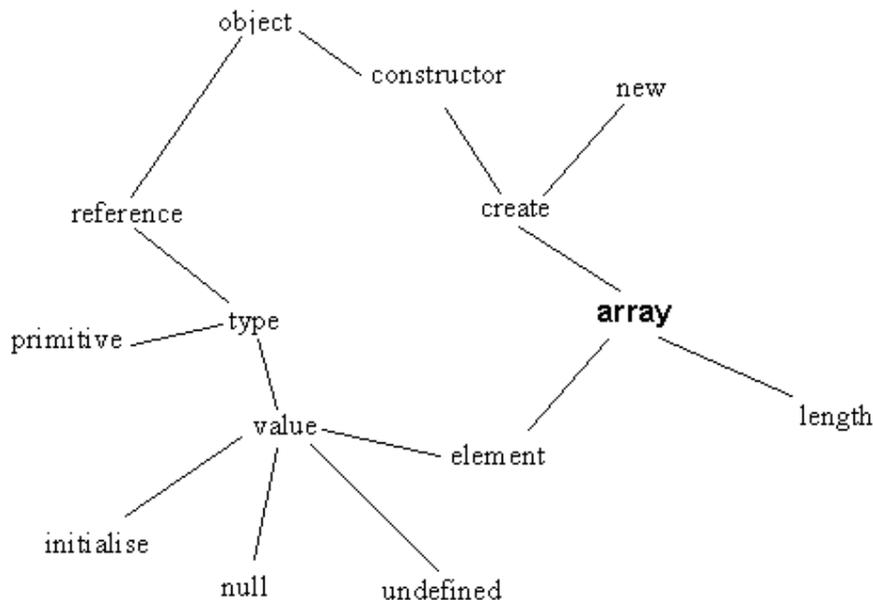
```
document.noise.src = images[ imageNumber ].src;
```

refers to the NAME of the image from the IMG tag, and resets the src property of this Image object to the appropriate array Image object src.

You can find a discussion of this activity at the end of the unit.

## 15.2.13 Arrays concept map

This figure illustrates the way different array concepts can be related.



## 15.3 The JavaScript object model

### 15.3.1 Introduction to the JavaScript object model

It is practically impossible to write useful JavaScript without using object properties and methods — even though many novice programmers do not realise they are making use of JavaScript objects.

Consider the following line of code:

```
document.write( "Sorry, that item is out of stock" );
```

The above line illustrates the use of many important object concepts, since it involves creating a new object ("Sorry..." is an instance built from the **String** constructor function), passing a reference to this new object as an argument to a method of another object (*write()* is a method of the document object).

JavaScript has a powerful and flexible object model. Unlike the Java programming language, JavaScript is a classless language: the behaviour and state of an object is not defined by a class, but rather by other objects (in the case of JavaScript, this object is called the object's *prototype*). While this major difference to languages such as Java and C++ often leads people to conclude that the JavaScript object model is a simple one, the JavaScript object model is powerful enough to allow a JavaScript programmer to extend its single inheritance abilities to allow for implementations of multiple inheritance, but also most of the functionality that JavaScript is considered to have "missing". All of this may also be done dynamically, at runtime — in other words, prototypes are defined while the JavaScript programme is executing, and can be updated with new methods and properties as and when they are required, and not only at compile time as with languages such as Java.

As you progress through this part of the unit, you will both learn new ways to use and define your own objects and constructor, and also you will come to more fully understand many JavaScript programming concepts you have learned up to now. The remainder of this introduction provides a more detailed overview of the JavaScript object model presented briefly in the first JavaScript unit.

### 15.3.2 Objects in JavaScript

A software system is considered to be a set of components — **objects** — that work together to make up the system. Objects have **behaviour** (defined by functions and methods) that carries out the system's work, and **state**, which affects its behaviour. As we previously mentioned, objects may be created and used by programming a **constructor function** to create objects, or **instances**. A constructor is merely a JavaScript function that has been designed to be used with the *new* keyword. The constructor defines what properties an object should have (the object's state), as well as its methods.

Programming an object involves declaring **instance variables** to represent the object's state, and writing **instance methods** that implement behaviour. The separate parts of an object's state are often called its **properties**. Thus, you might have a bank account object which had properties such as *holder* of account, *address* of holder, *balance* and *credit limit*; its methods would include the likes of *credit*, *debit* and *change address*.

JavaScript represents functions (which are used to implement methods, and to create objects) as data of type *Function* (notice the capital F), so in fact there is less difference between **data properties** and **functions** (or methods) than in most other object-based programming languages. Functions can be passed and stored in variables, and any function stored in a particular variable can be executed when required.

An object is said to **encapsulate** its state (often simply called its data) and its behaviour. In the case of JavaScript, this means that each object has a **collection** of properties and methods.

A JavaScript object can **inherit** state and behaviour from any other object. This object is called a **prototype**.

An object can obtain properties and methods from the constructor function from which the object is created; from its prototype, through inheritance; and from any properties and methods added separately to the object after it has been constructed (remember that Functions are just data that can be freely assigned to variables, as and when required). Two objects made from the same constructor function can have different properties and methods from each other, since extra properties and methods can be added to each object after they have been created.

Associated with inheritance is the separate concept of **polymorphism**, the ability to call a method on an object irrespective of the constructor function used to create the object, and expect the appropriate method to be called for the given object. For example, there can be two constructor functions, *Square()* and *Circle()*, which create objects representing squares and circles, respectively. The constructor can create these objects so that they each have an *area()* method that returns the area of the object. Of course, the method will operate differently for square and circle objects, but any variable holding either a square or a circle object can have the *area* method called on it, and polymorphism will ensure that the correct *area* method is called for the appropriate object.

### 15.3.3 Naming conventions

Although it can initially seem rather pedantic, the following naming convention is used throughout this unit, and is widely used in various programming standards. Using a standard convention helps a programmer to understand the code that they are reading, and you are advised to adopt this convention in your own programming.

All words of a **constructor** are spelt with an initial capital letter. Examples include:

- Document
- Form
- Triangle
- Array
- String
- BankAccount
- CreditCard

The first word in an **instance** (object) name is spelt with all lower case letters; all subsequent words making up the object name are spelt with an initial capital, as per usual. Examples include:

- Document.
- ageField.
- triangle1.
- nameArray.
- customerAccount1.

The first word of a **method** name is spelt in lower case; all subsequent words making up the object name are spelt with initial capitals. Examples include:

- area.
- setBalance.
- isGreaterThan.
- postTransaction.

The first word of a property is spelt in lower case; all subsequent words making up the object name are spelt with initial capitals. Examples include:

- triangle1.colour.
- width.
- firstName.
- addressLine1.
- Value.
- fontSize.

### Exercise 3

Which of the following are constructors:

- door1
- House
- MotorCar
- homeHampus

You can find a discussion of this Exercise at the end of the unit.

## 15.3.4 Creating objects and variables referring to objects

Objects are created in a number of ways, although the most common is by the use of the *new* operator. This is a unary, prefix operator, which means it is placed before a single operand. The *new* operator should be followed by a constructor function. Here are some examples:

```
new Object();  
var accountNumbers = new Array( 5 ); var firstName = new  
String( "Jose" );
```

### Where is the object created?

Sufficient free memory is located in which to create the new object. The *new* operator returns a **reference** to the location in memory where the new object has been created. In the first of the three examples above (*new Object();*), no variable is assigned the result of the object creation, therefore there is no way for the JavaScript programmer to refer to this object later in the programme. In almost all cases, a variable will be assigned the reference to the location of the newly created object.

In the second example above the **reference variable** *accountNumbers* is assigned the reference to the location of the newly created **Array** object.

### Note

Variables do not store objects, they store a reference to the object located elsewhere in memory.

## Creating String objects

Although the following way of creating **String** objects is perfectly acceptable JavaScript:

```
var firstName = new String( "Jose" );
```

it is usually much more convenient to use the special syntax provided for creating **String** literal objects. The same result could be achieved with the following statement:

```
var firstName = "Jose";
```

**String** objects can be created in two different ways because the creation of **String** objects is such a common feature of programming that the developers of the JavaScript language provided this second, simpler syntax especially for Strings.

## Creating Function objects

Just as the creation of Strings is so common the JavaScript developers provided a special syntax, the same has been done for make the creation of **Function** objects easier. As you will know from earlier units a function object can be created in the following way:

```
function myFunction( arg1,arg2, -, argN )
{
  //function statements go here
}
```

**Functions** are themselves objects (of the type Function), and can be created in the same way as objects using the **Function** constructor:

```
var myFunction = new Function( arg1, arg2, -, argN,
  "//function stateme
```

In most cases the former way of creating **Function** objects is more straightforward.

## 15.3.5 Object properties and methods

Objects have both data properties and methods (function properties). Often a property of one object is an object in its own right (or perhaps another kind of collection, such as an array). An example is the *forms* array property of the *document* objects — this property contains details (i.e. an array) for all the forms on a Web page.

### Properties

Variables (data) that "belong" to objects are called **properties**. In earlier units you have seen examples of properties, such as an array's *length* property:

```
var myArray = new Array( 5 );

alert( "length of array is: " + myArray.length );
```

In the example above the *length* property of the *myArray* object is accessed by the dot notation:

```
myArray.length
```

### Methods

Methods are object properties containing functions. In earlier units you have seen examples of methods, for example the *reverse* method of an array:

```
var myArray = new Array( 3 ); myArray[0] = 11;
myArray[1] = 22;
```

```
myArray[2] = 33;

alert( myArray.reverse() );
```

In the example above the *reverse()* method of the *myArray* object is invoked by the dot notation:

```
myArray.reverse()
```

As can be seen, the only difference from (data) properties is when methods are invoked with the use of the *()* (parentheses) operator — this is a special operator expecting a variable holding a Function object to the left, and optional operands (the function arguments) between the parentheses.

It is important to realise that methods are **properties**, but ones which can also be invoked since they are properties that hold Function objects.

## The 'dot' syntax

What is sometimes known as **dot** notation is used to indicate which properties an object owns. Ownership of properties can exist to any number of levels deep, and the dot notation can be used at every level. fine.

The use of the full stop (dot) can be thought of as meaning "the thing on the right belongs to the object (named collection) on the left". The general form for properties and methods to be invoked is as follows:

```
<object>.<property>
<object>.<method>()
```

Common examples include:

```
var myArray = new Array( 3 ) ; myArray.length;
var catpic = new Image();
catpic.src = "images/cartoons/sillycat1.gif";
```

In each of the above examples, the dot syntax is used to refer to a named piece of data (property) of an object. So we can refer to the *length* property belonging to the **Array** object *myArray*, and the *src* property of the **Image** object *catpic*.

We can see extensive use of the dot notation in the line:

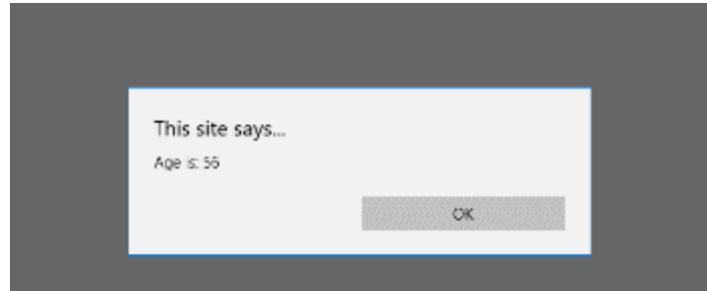
```
document.forms[0].age.value;
```

This line refers to the **value** property of the **age** property (itself an object) of the **form** object at index **0** of the **forms[]** array object of the **document** object. Code using such a reference to the value of a form's text input box is as follows:

```
<script>
    function showAge()
    {
        var newAge = document.forms[0].age.value; alert("Age is: " +
newAge );
    }
</script>
```

```
<form> Enter age:
<INPUT TYPE="text" NAME="age" SIZE=10>
<INPUT TYPE="button" NAME="button1"
VALUE="click me" SIZE=10 onClick="showAge()">
</form>
```

The browser output is:



## Variables are properties

When a JavaScript programme is being executed, a special object, called the 'global' object, is created. The global object exists until the programme terminates. All variables of a JavaScript programme are **properties** of this global object.

## Functions are methods

In the same way that all variables of a JavaScript programme are in fact properties of the global object, likewise all the functions of a JavaScript programme are **methods** of the global object.

## Functions are properties too

Since methods are just properties that hold Function objects, functions, being methods of the global object, are also stored in properties of the global object.

# 15.3.6 Some important JavaScript objects

Although you have been programming with objects for some weeks in JavaScript, you may not have been aware of some of the objects you have been working with. This section briefly describes important JavaScript objects.

You may wish to consult on-line and textual sources for further details of each JavaScript object.

## The 'global' object

When the JavaScript interpreter starts up, there is always a single **global** object instance created. All other objects are properties of this object. Also, all variables and functions are properties of this global object.

For client-side JavaScript, this object is the instance *window* of the constructor **Window**.

## The Window object for client-side JavaScript: window

When an HTML document is loaded into a browser, a single Window object instance, named *window*, is created. All other objects are properties of this window object. Since everything is a property of the window object, there is a relaxation of the dot notation when referring to properties. Thus each reference to a variable, function or object is not required to start with "window." — although this would be a more "accurate" notation to use, it is far less convenient.

Thus instead of writing:

```
document.write( "Hello" );
```

We could write:

```
window.document.write( "Hello" ); and so on.
```

One of the properties of the window object is the status property. This property is a String value that is displayed in the browser window's status bar. You can change the status bar with, or without, an explicit reference to the window object. Both these lines have the same effect:

```
window.status = "this is a test";  
status = "this is a test";
```

## The Document object: document

When an HTML document is loaded into a frame of a browser window, a Document object instance — named *document* — is created for that frame. This document object, like most objects, has a collection of properties and methods. Perhaps the most frequently used method of the document object is the *write()* method:

```
document.write( "sorry, that item is out of stock<p>" );
```

One useful property of the document object is the forms property. This is actually an array of Form objects with an element for each form defined in the document. So we could refer to the first form defined in the document as follows:

```
document.forms[0]
```

## The 'call' object

When a function (or method) is executed, a temporary object is created that exists for as long as the function is executing. This object is called the **call** object, and the arguments and local variables and functions are properties of this object.

It is through use of this call object that functions/methods are able to use local argument and variable/ function names that are the same as global variables/functions, without confusion. The call object is JavaScript's implementation of the concepts of variable/function **scoping** — i.e. determining which piece of memory is referred to by the name of a variable or function, when there are global and local properties with the same name.

## String objects

Strings are objects. A frequently used property of String is *length*. String includes methods to return a new **String** containing the same text but in upper or lower case. So we could create an upper case version of the **String "hello"** as follows:

```
var name = "hello";  
alert( name.toUpperCase() );
```

It should be noted that none methods belonging to string objects never change the string's value, but they may return a new String object, such as in the example above.

## Array objects

Since Arrays are rather an important topic in the own right, Array objects are given their own section at the end of this unit — although you may wish to skip ahead to read that section now to help your understanding of object with this more familiar example.

## Function objects

You should refer back to the earlier unit on functions.

As previously stated, functions are of the type Function, and can be treated as data variables or properties, or they can be treated as sub-programmes and executed using the () operator.

## Math objects

The Math object provides a number of useful methods and properties for mathematical processing. For example,

Math provides the following property:

```
Math.PI
```

that is useful for many geometric calculations.

An example of some other methods provided by the Math object include:

```
Math.abs( );  
Math.round( );  
Math.max( n1, n2 );
```

These may be used in the following way:

```
document.write( "<p>PI is " + Math.PI );
document.write( "<p>The signless magnitudes of the numbers -17 and 7 are "
    + Math.abs( -17 ) + " and " + Math.abs( 7 ) );
document.write( "<p>The interger part of 4.25 is " + Math.round( 4.25 )
);document.write( "<p>The larger of 17 and 19 is " + Math.max(17, 19) );
```

## 15.3.7 The 'Triangle' object

To focus our investigation of user defined constructors and objects, we will create 'Triangle' objects. We shall be defining the Triangle constructor function as follows:

- **Constructor Function** - *Triangle( initWidth, initHeight)*
- **Properties** - *numSides, width, height, colour*
- **Methods** - *larger( t1, t2), area()*

As you work through this section on JavaScript objects you will become familiar with not only with Triangle objects, but with the definition and use of your own constructors and objects.

The following is an example of some code using Triangles:

```
var triangle1 = new Triangle(10, 20);
document.write("<p><b>triangle1 has height: </b>" + triangle1.height );
document.write("<p><b>triangle1 has width: </b>" + triangle1.width );
document.write("<p><b>triangle1 has area: </b>" + triangle1.area() );
document.write("<p><b>triangle1 has colour: </b>" + triangle1.colour );
```

## 15.3.8 User defined objects

### Constructor functions

Objects are created with the new operator and a constructor function. The new operator requires the name of a function to its right (with optional arguments), and will return a reference to the memory location of the newly created object.

Examples of creating objects include:

```
var myArray1 = new Array( 3 ); var myArray2 = new Array();
var firstName = new String( "Jonathan" ); var catPicture = new
Image();
```

If we assume we have a defined a Triangle constructor, we can create new objects in just the same way as above:

```
var triangle1 = new Triangle( 5, 10 ); var triangle2 = new
Triangle( 10, 20 ); var triArray = new Array( 3 );
triArray[0] = new Triangle( 17, 92 );
```

A constructor function creates a new object, possibly initialising some of these new objects properties based on arguments provided when the function was called. We can see from the above that the Array() constructor function can create an array if provided with no arguments, or a numeric argument (for the size of the array), or a number of arguments (for the initial elements of the array).

Our Triangle constructor requires two arguments — the first is the width of the new triangle, the second is the height. Our Triangle constructor function looks as follows:

```
function Triangle( newWidth, newHeight )
```

```

    {
      this.width = newWidth; this.height = newHeight;
    }

```

Constructor functions make use of the special variable `this` — when a function is called with the `new` operator, a new object is created in memory, which the `this` variable refers to. Using this variable, the constructor is able to assign values to new properties called `width` and `height`. When a constructor function finishes, the reference to the newly created object is returned.

So after the following line is executed:

```
var triangle1 = new Triangle( 5, 10 );
```

the object `triangle1` should have a `width` property with value 5 and a `height` property with value 10. This can be investigated by using the dot notation and some `document.write()` statements:

```
document.write("<p> width is : " + triangle1.width );
document.write("<p> height is : " + triangle1.height );
```

We can imagine memory to be arranged something like the following (remember that all object variables are reference variables):

Memory location	value
<code>triangle1</code>	<code>001729</code>
<code>001727</code>	
<code>001728</code>	
<code>obj01.width</code>	<code>5</code>
<code>obj01.height</code>	<code>10</code>
<code>001731</code>	
<code>001732</code>	
<code>001733</code>	
<code>001734</code>	

## Functions are objects with properties

Now we have a constructor function, `Triangle`, we can use this function object as an object that defines `Triangles`.

### Object 'prototypes'

A constructor function object has a special property named `'prototype'` — this is an object through which inheritance is implemented for all objects created from the constructor: every object created from the constructor can inherit the properties and methods of the prototype object.

For example, by creating a new property or method for the prototype, this property or method is made available to all objects created from the constructor. Prototypes are a very useful way of defining methods for objects, and for setting default object properties.

For example, let us imagine that we wish all our **Triangle** objects to have the property `colour` initially set to the String `"blue"`. To provide this property to all **Triangle** objects created in the future, we can assign the property to the `Triangle` prototype as follows:

```
Triangle.prototype.colour = "blue";
```

Likewise, if we wish to make a method available to all `Triangle` objects, we need to assign a function to an appropriate property of the prototype. As useful method for `Triangle` objects is the calculation of their area. A function to calculate the area of a triangle could be defined as follows:

```
function triangleArea()
{
  // triangle height is half (width * height) var area = 0.5
  * this.height * this.width;
}
```

```

    // return calculation to 2 decimal places return
    Math.round( area * 100 ) / 100;
}

```

Notice how this function has been written to refer to whatever object it is called from by using the variable *this*.

We now need to assign this function to a property of the **Triangle** prototype. The method name *area* seems a good choice:

```
Triangle.prototype.area = triangleArea;
```

Notice, we are not executing this function, so we do not use the `()` operator. We are making `Triangle.area` refer to the same Function object as `triangleArea`.

After adding the above lines to a script, we can now see if a newly created `Triangle` object has a `colour` property and can use its `area` method:

```

var triangle1 = new Triangle( 5, 10 );

document.write("<p> colour property is: " +
triangle1.colour); document.write("<p> area method
returns: "+ triangle1.area() );

```

Remember, to invoke a function/method we must follow the function name with the `()` operator as above.

## 15.3.9 Implementing instance variables and methods

There is a distinction made to where variables and methods belong:

- **instance variables** — these are properties that can be different for each instance object (e.g. `triangle1` might have *height* 25, while `triangle2` has *height* 40).
- **instance methods** — methods that each object can apply to itself, and that has access to the properties of the object (e.g. `triangle1` can call its *area* method, this method returns a value calculated using the *height* and *width* properties of `variable1`).
- **class variables** — in a class-based object model, these are variables useful / relevant to the class as a whole, but that do not have anything to do with any particular instance of the class (e.g. the number of sides of a `Triangle` is 3, the value of `PI` is 3.1415926535). JavaScript, too, has the concept of "class variables", but the variables do not belong to classes, but rather to the constructor functions (which themselves are Function objects). These kinds of variables do not require an instance created from the `Triangle` or `Math` constructors in order for them to have meaning.
- **class methods** — Similar to class variables, these are methods that are useful / relevant to the constructor function, and does not require an instance of any object created by the constructor in order for it to be useful.

We shall briefly consider how each of these is implemented in JavaScript.

### Instance properties and instance methods

The previous section on the prototype property illustrates precisely what is meant by instance properties (properties) and instance methods:

- A property added to the prototype, or created inside the constructor, is an instance variable (property).
- A method added to the prototype (or in the constructor) is an instance method.

In fact, since a method is just a special kind of property, instance variables and instance methods can all be considered to be implemented as instance properties of the prototype.

You may find it useful to clearly comment the implementation of instance properties and methods in your code, such as the following lines illustrate:

```
////////////////////////////////////
// instance (object) methods
////////////////////////////////////

function triangleArea()
{

// triangle height is half (width * height) var area = 0.5 * this.height *
this.width;

// return calculation to 2 decimal places return Math.round( area * 100) /
100;
}

// add instance method "area" to the "prototype" property of "Triangle
// (i.e. add this method to the "prototype" property of the "Triangle"

Triangle.prototype.area = triangleArea;

////////////////////////////////////
// instance (object) variables (properties)
////////////////////////////////////
// the default colour for triangles is "blue" Triangle.prototype.colour =

"blue";
```

### **Class properties**

A "class properties" can be implemented by assigning a value to a property of the constructor object. This has been done for values such as *Math.PI*.

For example, we can implement the class property numSides with a value of 3 to the Triangle constructor as follows:

```
////////////////////////////////////
// class properties
////////////////////////////////////
// add property "numSides" Triangle.numSides = 3;
```

At any later point in the code we can then refer to this class property using the dot notation. For example:

```
document.write( "the number of sides of a triangle is: " +
Triangle.numSide
```

Class properties can be referred to even if no instances have been constructed (since the property belongs to the constructor, and not to the prototype).

### **Class methods**

A class method, such as *Math.abs()*, is implemented in the same way as a class property — a function is assigned to a property of the constructor function.

For example, we can we can implement a Triangle class method called larger(). This method requires two

arguments, each a Triangle object, and returns the String "first" if the first Triangle is the larger, otherwise the method returns "second" (i.e. if the second is the same or larger it returns "second"). First we must create a function:

```
function triangleLarger(t1, t2)
{
  if( t1.area > t2.area ) return "first";
  else
    return "second";
}
```

t1 and t2 are the two Triangle object references passed as arguments. This function makes use of the area instance method for each of the two Triangle objects — the larger triangle is defined to be the one with the larger area.

Then we must assign this function to an appropriately named property of the constructor object:

```
// add function to constructor (i.e. add to constructor function object

Triangle.larger = triangleLarger;
```

Notice, we are not executing this function, so we do not use the () operator. We are making Triangle.larger refer to the same Function object as triangleLarger.

The method can then be used as follows:

```
var triangle1 = new Triangle(10, 20);
var triangle2 = new Triangle(5, 10);
document.write("<p><b>the larger of triangles 'triangle1' and 'triangle
document.write( Triangle.larger( triangle1, triangle2 ) );
```

## 15.3.10 The JavaScript object search chain

When an object is created, a reference is created to the location in memory where the object's properties and methods are stored. Also, a reference is created to the object's prototype. When JavaScript wishes to retrieve the value of one of this object's properties, or to invoke one of this object's methods, it will first search the details stored for the object, and then if the desired property/object is not found it follows the reference to the object's prototype, where it continues to search there for the property or method. Again, if the property/method is not found in the prototype, it searches the prototype object's own prototype, and so on. Eventually, if not found, JavaScript will reach the object **Object** — this is the 'parent' of all built-in and user-defined objects.

If the property or method is not found, the value *undefined* is returned.

It is important to bear in mind that objects are **reference** types, as are prototype objects (since they are themselves only objects).

### Overriding inheritance

In the previous sections we have investigated the JavaScript features of prototypes and object inheritance. However, any object can override an inherited property or method by defining its own.

In our Triangle example, we have created a prototype *colour* property with a value of the **String** "blue". This means that, by default, all our **Triangle** objects will inherit this property and value. However, let us consider that while for most objects the colour blue is fine, for the object *triangleR* we may wish to have a *colour* property with value "red". The programming of this situation is very straightforward:

```
var t1 = new Triangle( 10, 20 );
var triangleR = new Triangle( 6, 6 ); triangleR.colour =
"red";
```

After execution of the above lines, part of the memory will look as follows:

Memory location		value
(Triangle) t1	001726	001729
	001727	
	001728	
obj01.width	001729	10
obj01.height	001730	20
	001731	
Triangle.prototype	001732	001734
	001733	
obj02.colour	001734	" blue"
obj02.area	001735	`` var area = 0.5 * this.height * this.width; return Math.round( area * 100) / 100;``
	001736	
	001737	
(Triangle) triangleR	001738	001741
	001739	
	001740	
obj03.width	001741	5
obj03.height	001742	10
obj03.colour	001743	" red"
	001744	

When JavaScript comes across code referring to the colour property of object *triangleR*:

```
document.write( "colour of triangleR is " + triangleR.colour );
```

it will follow the reference for *triangleR* to location 001741 and search for a property colour. It will succeed at location 001743. The value of this property will be returned, and this value, "red", will be passed to the *document.write()* method. Since the property value was found at the object referred to by *triangleR*, no search is made of the **Triangle** object *prototype*. In this way, object *triangleR* has been said to have overridden the inherited colour property with its own value of "red".

It is important to notice that although the object referred to by *triangleR* has overridden its inherited colour property, there has been no change to any of the other **Triangle** objects. So, for example, object *t1* still inherits the colour property of *Triangle.prototype*.

### Note - the difference between JavaScript and other object- language

In JavaScript, a specific object can override what it inherits from its object prototype. All other objects inheriting from the prototype can remain unaffected by this change. This differs from many other languages, such as Java, where any changes in inheritance are reflected across all objects of that given type.

### The dynamic nature of inheritance through prototype references

A final concept to appreciate regarding prototype based inheritance in JavaScript is that the inheritance of prototype properties and methods is dynamic: if an object is created and the prototype is subsequently changed, the changes will be reflected in the state and behaviour of any object inheriting from it.

We shall illustrate this dynamic inheritance with the following code:

```
var myTriangle = new Triangle( 6, 8 ); var triangle2 = new
Triangle( 11, 22 );
document.write("<p> colour of myTriangle is " +
myTriangle.colour ); document.write("<p> colour of triangle2
is " + triangle2.colour );
Triangle.prototype.colour = "green";
document.write("<p> colour of myTriangle is " +
myTriangle.colour ); document.write("<p> colour of triangle2
is " + triangle2.colour );
```

After the execution of the first two lines:

```
var myTriangle = new Triangle( 6, 8 ); var triangle2 = new Triangle( 11, 22 );
```

part of the memory will look as follows:

Memory location	value
(Triangle) myTriangle	001726
	001727
	001728
obj01.width	001729
obj01.height	001730
	001731
Triangle.prototype	001732
	001733
obj02.colour	001734
obj02.area	001735
	001736
	001737
(Triangle) triangle2	001738
	001739
	001740
obj03.width	001741
obj03.height	001742
	001743
	001744

The diagram shows a table of memory locations and values. Arrows on the right side of the table point from object instances to their prototype objects. Specifically, arrows point from (Triangle) myTriangle to Triangle.prototype, from (Triangle) triangle2 to Triangle.prototype, and from obj03.width and obj03.height to obj03.width and obj03.height respectively.

So when the first two document.write() statements are executed:

```
document.write("<p> colour of myTriangle is " + myTriangle.colour );
document.write("<p> colour of triangle2 is " + triangle2.colour );
```

JavaScript retrieves the inherited colour "blue" for each object. However, after execution of the line that changes the prototype property:

```
Triangle.prototype.colour = "green";
```

memory is changed to look as follows:

Memory location	value
(Triangle) myTriangle	001726
	001727
	001728
obj01.width	001729
obj01.height	001730
	001731
Triangle.prototype	001732
	001733
obj02.colour	001734
obj02.area	001735
	001736
	001737
(Triangle) triangle2	001738
	001739
	001740
obj03.width	001741
obj03.height	001742
	001743
	001744

The diagram shows the same memory layout as before, but with the value of obj02.colour changed to "green". The other values remain the same. Arrows on the right side of the table point from object instances to their respective prototype objects.

Thus when the last two `document.write()` statements are executed, the reference is followed to the `colour` value at location 001734 "green", and it is this changed property value that is inherited by both the **Triangle** objects.

It is important to understand, then, that **changing a prototype dynamically changes the inherited properties and methods for all objects sharing the prototype from that point onwards**, and should not be an action used lightly, or when some simpler solution is possible.

## 15.3.11 Object activities

### Activity 4 - Iterating through object properties

Create an object `customer1` with the following properties and values:

Object — <code>customer1</code>	
Property name	Property value
<code>name</code>	<code>"fred"</code>
<code>age</code>	<code>29</code>
<code>creditLimit</code>	<code>500</code>

Write a `for/in` loop that will iterate through the properties of this object, displaying each property name and value a separate line.

The browser output should look something like the following:



You can find a discussion of this activity at the end of the unit.

### Activity 5 — Constructor function for Rectangle

Create a constructor function for **Rectangle** objects. The properties of new **Rectangle** objects to be initialised in the constructor are its `length` and `height`.

Class — <code>Rectangle</code>	
	<b>Constructor function</b> <code>Rectangle( initLength, initHeight )</code>
<b>Instance Properties</b>	<b>Instance Methods</b>
<code>length</code>	
<code>height</code>	

Your constructor function should be designed to create new **Rectangle** objects with code such as the following (note the `length` is the first argument, and the `height` the second):

```

var rect1 = new Rectangle( 15, 20 ); document.write("<p> length is : " +
rect1.length ); document.write("<p> height is : " + rect1.height );
  
```

You can find a discussion of this activity at the end of the unit.

## 15.4 Arrays as objects

Arrays are objects, and have properties and methods like any other object. The prototype of the array object can be illustrated by the following figure:

Class — Array	
Instance Properties	Instance Methods
<code>length</code>	<code>join()</code>
	<code>sort()</code>
	<code>reverse()</code>
	<code>concat()</code> recent addition to JavaScript
	<code>slice()</code> recent addition to JavaScript
	<code>splice()</code> recent addition to JavaScript
	<code>push()</code> recent addition to JavaScript
	<code>pop()</code> recent addition to JavaScript
	<code>shift()</code> recent addition to JavaScript
	<code>unshift()</code> recent addition to JavaScript
	<code>toString</code> recent addition to JavaScript
	<code>toSource()</code> recent addition to JavaScript

Earlier this unit we dealt in detail with the *length* property of arrays. The first three methods of the **Array** object list above are discussed below. You are advised to consult on-line sources and books for details of the many other **Array** functions.

### 15.4.1 Array method: join

The *join()* method is a straightforward way of converting from the tabular, numerically indexed structure of an **Array** to a simple text **String**. *join()* copies all elements of an **Array** into a **String** object, separating each element value with the provided String argument, or with a comma "," if no argument is provided.

The general form of the *join()* method is:

```
<array>.join( <separator> );
```

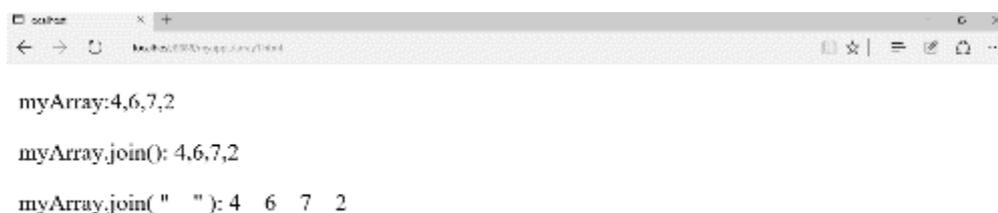
where <array> is a variable holding an array, and <separator> (if present) is a String to be used to separate each array element. The *join()* method is called by default when an **Array** object is passed as a parameter to a *document.write(...)* statement.

An example of some code using the *join()* method is:

```
var myArray = new Array( 4, 6, 7, 2 );

document.write("<p> myArray:" + myArray );
document.write("<p> myArray.join(): " + myArray.join() );
document.write("<p> myArray.join( \" _ \" ): ' + myArray.join( \" _ \" ) );
```

The output of the code is:



As can be seen from the output, calling *document.write(...)* with simply the **Array** variable as an argument is the

same as providing the **Array** name and its default *join()* method. However, when the *join()* method is called with the argument " \_ " this String is returned as the separator for each array element in the String returned by the method.

## 15.4.2 Array method: sort

As its name suggests, the *sort()* Array method provides a mechanism for changing the order of the elements of an **Array** according to provided criteria.

*sort()*'s default ordering is ascending, alphanumeric order. This produces some unintuitive results for numeric values:

```
var myArray = new Array( 4, 16, 7, 2);
document.write("<p> myArray: " + myArray );
document.write("<p> myArray.sort(): " + myArray.sort() );
```

Since 16 comes **alphabetically** before 2 (1 is alphanumerically lower than 2 ), it appears first in the sorted array. To sort array elements according to numerical (or some other) order an ordering function has to be provided as an argument to the *sort()* method. This ordering **function** must accept two arguments and return a numeric indication indicating which of the two arguments should occur before the other. If the function returns the value N, then:

- N < 0: if the first argument should appear before the second.
- N > 0: if the second argument should appear before the first.
- N = 0: if the two arguments are equivalent.

An example of a function that places elements into ascending numeric order is:

```
function smaller(n1, n2)
{
    return (n1 - n2);
}
```

The above function *smaller()* returns a negative value if the first argument is the smaller, zero if the two numbers are equal, and a positive number if the second argument is the smaller.

A more explicit (although less elegant) function, this time to return the larger of two numbers, can be written as follows:

```
function larger(n1, n2)
{
    if( n1 > n2)
        return -1; // "n1" first else if (n1 < n2)
        return 1; // "n2" first else
        return 0;
}
```

As we can clearly see, this *larger()* function returns -1 if the first argument is larger, 1 if the second argument is larger, and zero if the arguments are the same. An example of some code that illustrates both the default (alphabetical) sorting, and sorting using the *larger()* function above, is as follows:

```
var myArray = new Array( 4, 16, 7, 2);
document.write("<p> myArray: " + myArray );
document.write("<p> myArray.sort(): " + myArray.sort());

////////////////////////////////////
// function to return the smaller of 2 numbers
//////// so will order numbers in DESCENDING order

function larger(n1, n2)
{
    if( n1 > n2)
```

```

return -1; // "n1" first else if (n1 < n2)
return 1; // "n2" first else
return 0;
}
document.write("<p> myArray.sort(): " + myArray.sort(larger));

```

The browser output from the above code is:



### 15.4.3 Array method: reverse

The *reverse()* method does not return any values; rather, it rearranges the **Array** elements into reverse order (i.e. reverse order of the numerical indices from 0..(length-1)).

Consider an array *myArray* with the following values:

array <i>myArray</i>	
Index	Value
<i>myArray</i> [0]	"hello"
<i>myArray</i> [1]	false
<i>myArray</i> [2]	16

If the *reverse()* method is invoked on *myArray*, the following is obtained:

array <i>myArray</i>	
Index	Value
<i>myArray</i> [0]	16
<i>myArray</i> [1]	false
<i>myArray</i> [2]	"hello"

Some code to demonstrate this is as follows:

```

var myArray = new Array(3);

myArray[0] = "hello";
myArray[1] = false;
myArray[2] = 16;

document.write("<p> myArray: " + myArray );

myArray.reverse();

document.write("<p> after reversing...");

```

```
document.write("<p> myArray: " + myArray );
```

The browser output of the above code is:



## 15.4.4 Single and multi-dimensional arrays

The discussion of arrays have so far concerned what are known as single-dimension arrays, which are arrays forming an arrangements of values that can be accessed using a single numeric index, and can be pictured as a row of boxes, each box being numbered (by the index) and holding a value. Most programming languages provide facilities for multi-dimensional arrays, which are arrays requiring multiple numeric indices. A two dimensional array can be pictured as a book shelf, where each shelf is itself a single-dimension array.

In JavaScript, a two-dimensional array — holding months and the days of the month — can be created as follows:

```
var delivery = new Array(12); delivery[0] = new Array(31);  
  
// January  
delivery[1] = new Array(29); // allow for Feb in leap years  
delivery[2] = new Array(31); // March...  
delivery[11] = new Array(31); // December
```

Notice that a two dimensional array is created by setting each element of an single-dimensional array to be arrays themselves.

This array could be used to represent the days on which a company could possibly perform a delivery, and we could place true/false values into each element to indicate whether a delivery is possible on that day or not. If a delivery is not possible on the 1st or 2nd of January, but is possible on the third, we might write:

```
delivery[0][0] = false; // no delivery possible 1st Jan  
delivery[0][1] = false; // no delivery possible 2nd Jan  
delivery[0][2] = true; // delivery is possible 3rd Jan
```

However, let us assume that due to poor climate and annual staff holidays that the company cannot perform deliveries during August. JavaScript's flexible nature allows us to do the following:

```
delivery[7] = false; // August - no deliveries
```

Notice that a multi-dimensional array can easily have different sizes for its different rows, and some rows need not even be arrays at all.

To make working with multi-dimensional arrays easier, one can either add new methods to the Array prototype, or create new constructor functions which use Array as their prototype. Consider the following constructor function which attempts to make creating multi-dimensional arrays easier. It takes two arguments — the size of the first and second dimensions — and creates a two dimensional array:

```
function two_dim_array(length_d1, length_d2)  
{  
    // loop to make each element of "this" an  
    // array with length "length_d2"  
  
    var i = 0;  
    while( i < length_d1 )
```

```

        {
            this[i] = new Array( length_d2 );
            i++;
        }

// make the "length" property of array "this"
// an object with properties
// "d1" = "length_d1"
// "d2" = "length_d2"

this.length = { d1:length_d1, d2:length_d2 };
}
two_dim_array.prototype = Array;

var myArray = new two_dim_array(3,2);

```

## 15.5 Objects as associative arrays

While it is clear that arrays are JavaScript objects, JavaScript also allows objects and their properties to be accessed as if they were arrays. Normal JavaScript arrays index their values in order using integer indices. When treating JavaScript objects as arrays, Strings are used as indices to associate each property value with the object name. Arrays of this kind are called "associative arrays".

To refer to an object property using array notation, simply pass the property name as a String to the array square brackets applied to the object, as follows:

```
objectName[ "propertyName" ]
```

An example of the use of associative array notation for accessing object properties is illustrated in the following code:

```

var object1 = new Object;
object1.name = "Joanne"; object1.age = 27;
object1.nationality = "British";
document.write(<p> property name: " + object1["name"] );
document.write("<p> property age: " + object1["age" ] );
document.write("<p> property nationality: " +
object1["nationality" ] )
var myArray = new Array(5);
document.write("<p> array length: " + myArray["length" ] );

```

It is important to note that although the square bracket indexing is used with associative arrays, it is objects that are being processed, and that these objects must be created with constructor functions (and the *new* keyword), or with object literals — associative arrays are not created using the built-in Array object.

### 15.5.1 Enumerating associative arrays with FOR/IN loop statements

One form of loop statement created for the processing of associative arrays is the FOR/IN statement. This statement allows the processing of all user-defined properties of arrays (and all inherited user-defined properties).

An example of the FOR/IN statement is:

```

var object1 = new Object;

object1.x = 29;
object1.y = 6;
object1.z = 55;

for( var property in object1)

```

```
{
  document.write("<p>the value of property " + property + "
  is " + objec
}
```

## Note

It is important to note that the developers of JavaScript did not prescribe the order in which the FOR/IN statement processes object properties. Therefore the output of such statements may vary from browser to browser, and the ordering cannot be relied upon to always be the same.

If the order is important, either more sophisticated programming is required, or the code must be tested on the specific browsers the JavaScript needs to run on — along with a clear warning that the code may not work on other browsers. Obviously, it is best to use this statement in such a way that the order in which properties are processed does not matter.

The real usefulness of the FOR/IN statement is that it allows the iterative processing of all properties of an object, in the same way that a numeric loop variable can be used to iteratively process all elements of a normal array. This technique is mainly useful where

- The order of the properties either does not matter, or can be resolved
- All (or at least most) properties of an object need to be processed in the same way
- There are a large number of properties, so that a FOR/IN statement saves both programme statements and helps avoid the chance of forgetting to process an object property

## 15.5.2 Using FOR/IN for numerically index arrays

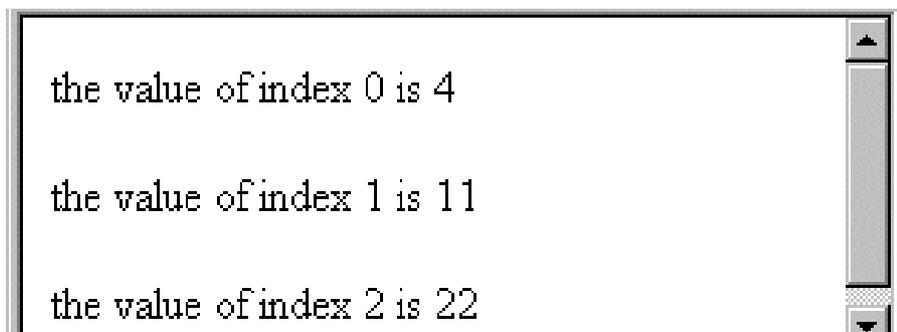
FOR/IN statements can be used to enumerate normal, numerically index arrays. For example, consider the code:

```
var object1 = new Array( 3 );

object1[0] = 4;
object1[1] = 11;
object1[2] = 22;

for( var index in object1)
{
  document.write("<p>the value of index " + index + " is " +
  object1[ index])
}
```

The above code produces the following output on a browser:



However, as with any use of FOR/IN, there is no guarantee that the elements will be retrieved in the numerical index order. To ensure that elements are enumerated in numerical sequence (starting at 0), use a loop such as the following:

```
var object1 = new Array( 3 ); object1[0] = 4;  
object1[1] = 11;  
object1[2] = 22;
```

```

var index = 0;

while( index < object1.length )
{
    document.write("<p>the value of index " + index + " is " +
    object1[ i

    index++;
}

```

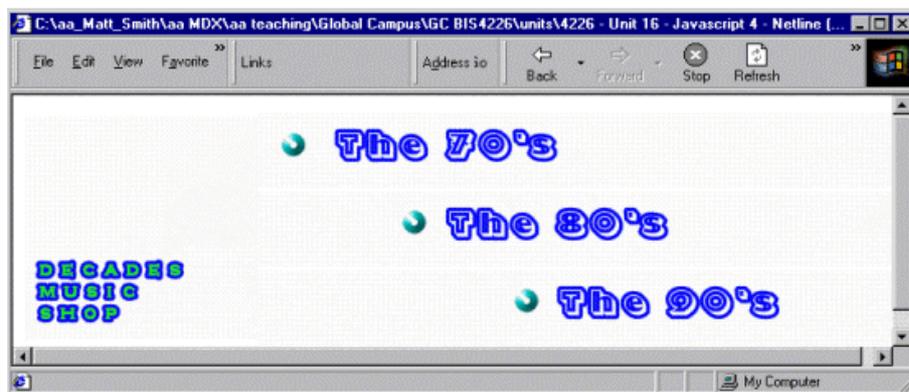
**Note**

You may wish to read about the more elegant FOR loop statement for simple numerical loops such as the one above.

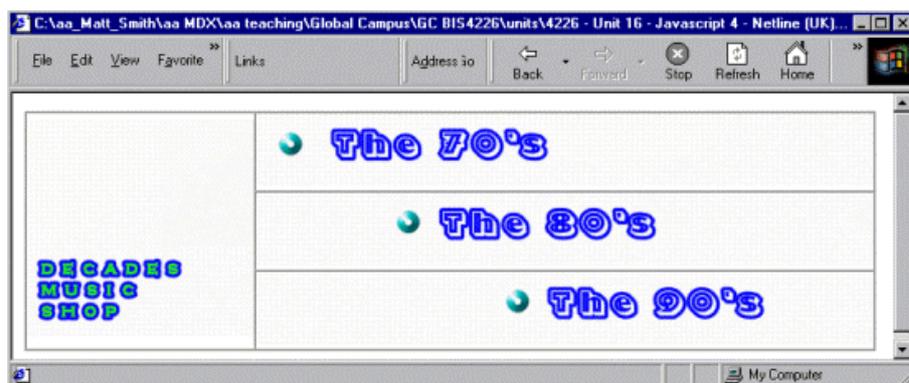
## 15.5.3 Activities and further work

### Activity 6 — Using Arrays of Images to improve "Decades Music Shop"

The company "Decades Music Shop" has set up an on-line music ordering website. The site looks as follows:

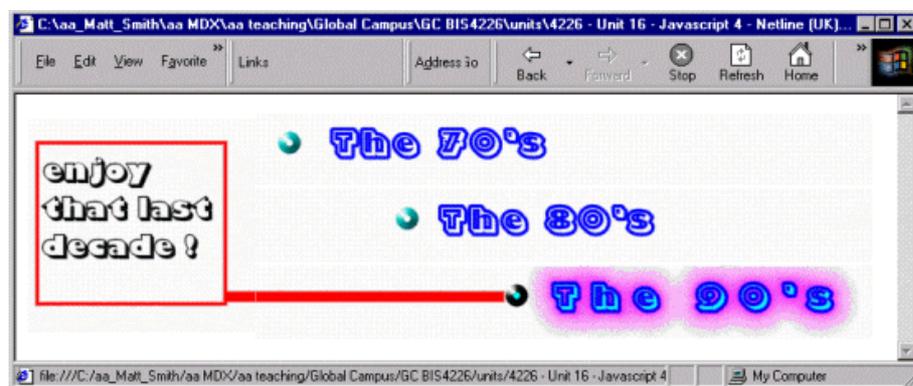
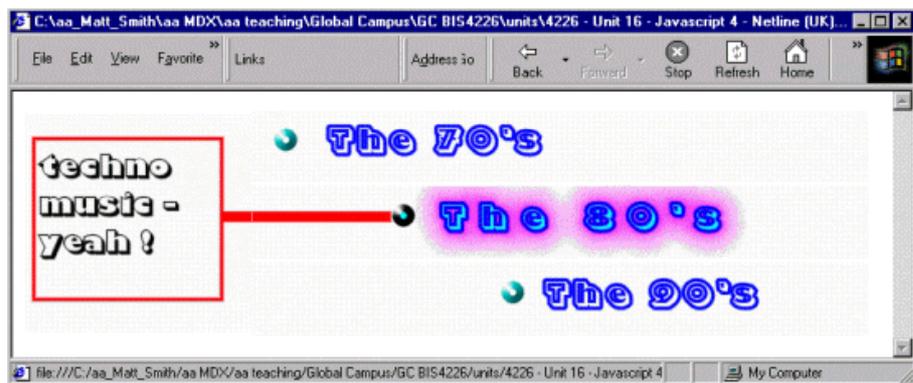
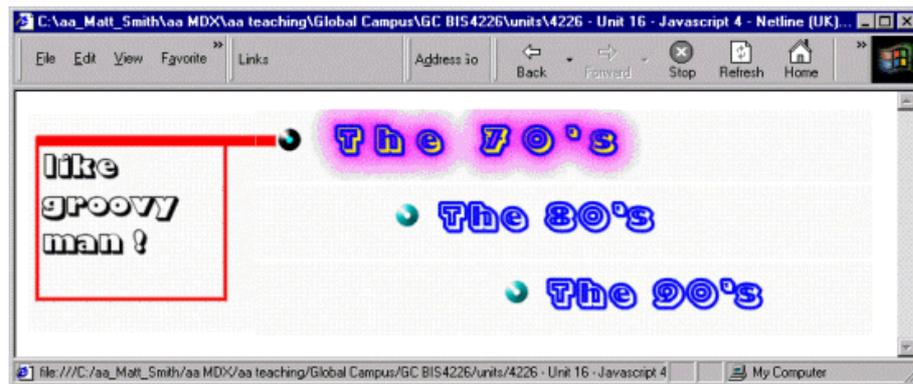


The Webmaster who set up the site has made this a visually interactive home page by arranging the four images in a table. The table has three rows for "The 70s", "The 80s" and "The 90s" and the first column is an image spanning all 3 rows. The same page, with the table border set to 1 is as follows:



The images for "The 70s", "The 80s" and "The 90s" change when the mouse is over them — i.e. they have been made rollover images via the onmouseover and onmouseout event

attributes of the IMG tag. The three screenshots below show how the page changes when the mouse is over each of these images in turn:



The current music shop Webmaster does not know about Arrays, and so has implemented a separate JavaScript function for each images onMouseOver and onMouseOut event — i.e. the Webmaster has had to create two functions for each image. The functions for the 70s image are as follows:

```
function select70s()
{
    document.the70s.src = "the70s_selected.gif";
    document.textArea.src = "the70s_selected_TEXT.gif";
}

function deselect70s()
{
    document.the70s.src = "the70s_UNselected.gif";
    document.textArea.src = "UNselected_TEXT.gif";
}
```

```
}
```

The HTML table is created with the following code:

```
<TABLE CELLSPACING=0 CELLPADDING=0 WIDTH=202 BORDER=0>

<TR>
  <TD rowspan = 3>
    <IMG NAME="textArea" SRC="UNselected_TEXT.gif">
  </TD>

  <TD>
    <A HREF="the70s.html" onMouseOver="select70s();"
      onMouseOut="deselect70s();" >
    <IMG NAME="the70s" SRC="the70s_UNselected.gif"
      BORDER=0></A>
  </TD>
</TR>

<TR>
  <TD>
    <A HREF="the80s.html" onMouseOver="select80s();"
      onMouseOut="deselect80s();" >
    <IMG NAME="the80s" SRC="the80s_UNselected.gif"
      BORDER=0></A>
  </TD>
</TR>

<TR>
  <TD>
    <A HREF="the90s.html" onMouseOver="select90s();"
      onMouseOut="deselect90s();" >
    <IMG NAME="the90s" SRC="the90s_UNselected.gif"
      BORDER=0></A>
  </TD>
</TR>

</TABLE>
```

Notice how the 70s image *onMouseOver* and *onMouseOut* event attributes call the two functions listed above.

Your task is to reduce the number of required functions from six to two. You are to create a single function called `selectImage()` and another called `deselectImage()`, which are passed the number of the image to select or deselect. Refer to the 70s image as number 0, the 80s image as number 1 and the 90s image as number 2.

You are to achieve this reduction of functions by creating three arrays, called `selectedImages`, `deselectedImages`, and `imageNames`. They contain Strings holding the name of the relevant image as stated in the HTML `IMG NAME="..."` tag. Each array should contain an `Image` object whose `src` property has been set to the appropriate image.

changing a document image when referring to an array can be done in a way similar to this:

```
document.the80s.src = selectedImages[1].src;
```

If your function calls its index argument `imageNumber`, you can create a String that contains the statement you wish executed, and then use the `eval()` function to execute that String:

```

var imageStatement = "document." + imageNames[imageNumber ] ;

imageStatement += ".src = selectedImages[" + imageNumber + "].src";

eval( imageStatement );

```

So if imageNames[ imageNumber ] contains "the80s" and then the variable imageStatement will contain precisely the statement you wish to execute!

You can find a discussion of this activity at the end of the unit.

## Activity 7 — Using associative arrays of images to improve "Decades Music Shop"

Associative arrays offer a way to associate names rather than numbers with array elements.

Your task is to make the music\_arrays.html more meaningful by replacing your arrays of images with objects that have a property for each image. This means that the need for an array of image names is removed, the need for an eval statement is removed, and the argument to the select and deselect functions becomes a more meaningful String, such as "the70s", or whatever IMG NAME has been set to.

HINT 1: Create an associative array (object) with images as properties.

Replace your array creation statements with object creation. Then replace array element assignment statements with object property statements:

```

var selected = new Object; selected.the70s = new Image();
selected.the70s.src = "the70s_selected.gif";

```

Thus we now have an object (i.e. an associative array) named selected, that can have its Image properties accessed via the property name, e.g.:

```

selected["the70s"].src;

```

HINT 2: Pass the property name as an argument from onMouseOver and onMouseOut Replace your

HTML IMG lines with something similar to the following:

```

<A HREF="the70s.html" onMouseOver="selectImage('the70s');"
  onMouseOut="deselectImage('the70s');">

<IMG NAME="the70s" SRC="the70s_UNselected.gif" BORDER=0>

</A>

```

(remember, since you cannot have double quotes within double quotes, you will need to mix double and single quotes for the onMouseOver JavaScript one-liner String, since you are passing a String argument as part of your JavaScript statement).

You can find a discussion of this activity at the end of the unit.

## Activity 8 — Using user-defined object with methods to improve "Decades Music Shop"

The previous activity greatly reduced the amount of code, and improved the readability and meaningfulness of function calls.

However, at present we have three objects containing our images, and two isolated functions. We can tidy things up even further by encapsulating the select and deselect functions into user-defined, rollover image objects.

Create a constructor function called *RolloverImage()* as follows:

Class — <code>RolloverImage</code>	
	<b>Constructor function</b> <code>RolloverImage (</code> <code>    nameString,</code> <code>    selectedIMG,</code> <code>    unselectedIMG,</code> <code>    selectedTEXT,</code> <code>    unselectedTEXT )</code>
Instance Properties	Instance Methods
<code>name</code>	<code>select ()</code>
<code>selected</code>	<code>deselect ()</code>
<code>unselected</code>	
<code>selectedTEXT</code>	
<code>unselectedTEXT</code>	

Where the instance properties are as follows:

String	name	the name of the image this
Image	selected	the selected version of the
Image	unselected	the unselected version of the
Image	selectedTEXT	the selected version of the
Image	unselectedTEXT	the unselected version of the TEXT IMG (in fact this is the same for

Create the instance methods for the prototype — these should be the methods *select()* and *deselect()*.

Create an object in which to store your RolloverImage objects, call this object *rolloverImages*.

Create, as properties of your object *rolloverImages*, three instances of RolloverImage named *the70s*, *the80s* and *the90s*. This lets you use statements such as:

```
rolloverImages.the70s.select()
```

for `onMouseOver` events, and so on.

HINT: The HTML table entries should call object methods.

Your HTML table entries should look something like the following:

```
<TD>

  <A HREF="the70s.html "
  onMouseOver="rolloverImages.the70s.select()" onMouseOu
  <IMG NAME="the70s"
  SRC="rolloverImages.the70s_UNselected.gif" BORDER=0>
  </A>

</TD>
```

You can find a discussion of this activity at the end of the unit.

## 15.6 Review Questions

1. What is an array?

You can find the answer end of the unit.

2. What is the role of the number 4 in the line:

```
Alert( myArray[4] );
```

You can find the answer end of the unit.

3. How many elements, and what are their indices, in the array `ages` created in the statement:

```
var ages = new Array(3)
```

You can find the answer end of the unit.

4. Consider this code working with numeric variables:

```
var a1 = 5; var b1 = a1;

document.write( "<p> a1 = " + a1 ); b1 = 7;

document.write("<p> a1 = " + a1 );
```

and this code working with array variables

```
var a2 = new Array( 2 );

a2[0] = 5;
a2[1] = 6;
var b2 = a2;
document.write("<p> a2 = " + a2 ); b2[0] = 7;

document.write("<p> a2 = " + a2 );
```

Browser output when this code is executed is as follows:

---

```
a1 = 5  
  
a1 = 5  
  
a2 = 5,6  
  
a2 = 7,6
```

Why is it that a2 is changed when b2 is changed, but a1 is not changed when b1 changes? You can find the answer end of the unit.

5. Why does document start with a lower case in the statement: `document.write("sorry, product out of stock");` You can find the answer end of the unit.
6. What output will the following code produce — try to work this out on paper, don't enter the code and try it out yet!

```
<script src="triangle_class.js"></script>  
  
<script>  
  
    var triangle1 = new Triangle(10, 20); var t2 = triangle1;  
  
    t2.width = 5;  
  
    document.write("triangle1 has width: " + triangle1.width );  
  
</script>
```

You can find the answer end of the unit.

7. Are JavaScript objects just data structures? You can find the answer end of the unit.

## 15.7 Discussion Topics

1. An object is just an array indexed by property names.

You can some contribution to this topic at the end of the unit.

2. Why should programmers try to avoid changing prototypes in the middle of code where objects using the prototype are being created and used?

You can some contribution to this topic at the end of the unit.

## 15.8 Answers

## 15.8.1 Discussions of Exercise 1

The answers and discussion of the questions are as follows:

An element of an array is referred to with the array name and the index in square brackets. The first element is that indexed with 0, so it is `weekDays[0]` and so on:

- The first element is: `weekDays[0]`
- The last element: `weekDays[6]`
- The 4th element: `weekDays[4]`
- The value of `weekDays[0]` is "Monday" (we can see this from the figure), and so on.
- The value of the first element: "Monday"
- The value of the 4th element: "Friday"
- The element of the array is referring to the part of `weekDays` that contains the value, i.e. `weekDays[0]` or `weekDays[6]` and so on.
- The element containing String "Monday": `weekDays[2]`
- The element containing String "Saturday": `weekDays[5]`

The index of the array element is referring to the number of the `weekDays` element containing the given value; since "Monday" is stored in `weekDays[2]`, the index of this element is 2:

- The index of the element containing String "Monday": 2
- The index of the element containing String "Saturday": 5

## 15.8.2 Discussions of Exercise 2

After execution of the first two lines, memory looks as follows:

Memory location	value
<code>myArray</code>	<code>002803</code>
<code>002802</code>	
<code>[0]</code>	<code>002803</code>
<code>[1]</code>	<code>002804</code>
<code>[2]</code>	<code>002805</code>
<code>[3]</code>	<code>002806</code>
	<code>002807</code>
<code>thing2</code>	<code>002803</code>
<code>002809</code>	

Since `thing2` refers to the same array values as `myArray`, after the three lines:

```
thing2[1] = 27;  
thing2[2] = 19;  
thing2[3] = 77;
```

memory will have been changed as follows:

Memory location	value
myArray	002803
002802	
[0]	002803
[1]	002804
[2]	002805
[3]	002806
	002807
thing2	002803
002809	

Therefore the value of *myArray[2]* is now 19.

### 15.8.3 Discussions of Activity 1

The following code will fulfil the requirements:

```
<html> <script> <!--
var ages = new Array( 21, 14, 33, 66, 11 );
document.write( "<p><b>ages[0] </b>" + ages[0]);
document.write( "<p><b>ages[1] </b>" + ages[1]);
document.write( "<p><b>ages[3] </b>" + ages[3]);

// --> </script> </html>
```

This first line:

```
var ages = new Array( 21, 14, 33, 66, 11 );
```

creates an array, and initialises it with the 5 values. This could have also been achieved as follows:

```
var ages = new Array( 5 ); ages[0] = 21;
ages[1] = 14;
ages[2] = 33;
ages[3] = 66;
ages[4] = 11;
```

The *document.write( ... )* lines display the values of each specified array element.

### 15.8.4 Discussions of Activity 2

To create the primes array we can write the following:

```
var primes = new Array( 6 );
```

To put the values into the array we can write the following:

```
primes[0] = 1;
primes[1] = 2;
primes[2] = 3;
primes[3] = 5;
primes[4] = 7;
primes[5] = 11;
```

Alternatively we could create and initialise the array all in a single line:

```
var primes = new Array( 1, 2, 3, 5, 7, 11 );
```

To iterate through the array we need an index variable (we'll use *i*) and a while loop that will keep looping

while *i* is less than the value of the length property of the array. By using the length property in our loop condition we make the loop general, in that it will loop through the whole array no matter the length of the array.

```
var i = 0;
while (i < primes.length)
{
    document.write("<p> " + primes[i] );
    i++;
}
```

## 15.8.5 Discussions of Exercise 3

Constructor functions are named with an initial capital letter, so the following are the names of constructor functions:

- House
- MotorCar

The others may be objects, or methods, or property names:

- door1
- homeHampus

## 15.8.6 Discussions of Activity 3

An example of an HTML file resulting in the desired animation could be the following:

```
<HTML> <HEAD> <SCRIPT> <!--
// function to:
// display image
// increment image number
// increase delay time by 1/4 second
// if not last image then delay and call self
function animate()
{
    document.noise.src = images[ imageNumber ].src;
    imageNumber++;
    delay += 250;

    if( imageNumber < 4 )
        setTimeout( "animate()", delay );

    // if imageNumber = 4 the animation has finished
    // and this function can terminate
}

// function to:
// set imageNumber for "taptap" image
// set delay to 1/4 second (250 milliseconds)
// execute the animate() function
function
startAnimation()
{
    imageNumber = 0;
```

```
delay = 250; animate();  
}
```

```

// set up image array
var images = new Array( 4 );

images[0] = new Image(); images[0].src = "taptap.gif";
images[1] = new Image(); images[1].src =
"knockknock.gif"; images[2] = new Image(); images[2].src
= "BANGBANG.gif"; images[3] = new Image(); images[3].src
= "silence.gif";

// declare the global variables for the imageNumber and
the delay duration
var imageNumber;
var delay;

//--> </SCRIPT> </HEAD>

<BODY>

<IMG NAME="noise" SRC="silence.gif"
onMouseOver="startAnimation()">

</BODY> </HTML>

```

It is important you understand this programme before moving on the later object activities.

## 15.8.7 Discussions of Activity 4

The object *customer1* can be created as follows:

```
var customer1 = new Object();
```

The properties of this object can be created and assigned as follows:

```
customer1.name = "fred"; customer1.age = 29;
customer1.creditLimit = 500;
```

A *for/in* loop to iterate through and display the properties on separate lines could be:

```
for( var prop in customer1 )
{
  document.write( "<p><b> property </b>" + prop + "<b> has
value </b> " + customer1[prop] )
}
```

## 15.8.8 Discussions of Activity 5

Your constructor function must be named *Rectangle*. It should take two arguments, named along the lines of *newLength* and *newHeight*. The function should assign the value of the first argument to *this.length*, and the value of the second argument to *this.width*:

```
function Rectangle( newLength, newHeight )
{
  this.length = newLength; this.height = newHeight;
}
```

## 15.8.9 Discussions of Activity 6

Your code should set up the three image arrays:

```
var selected = new Array(3); selected[0] = new Image();
selected[0].src = "the70s_selected.gif"; selected[1] =
new Image(); selected[1].src = "the80s_selected.gif";
selected[2] = new Image(); selected[2].src =
"the90s_selected.gif";

var unselected = new Array(3); unselected[0] = new
Image();
unselected[0].src = "the70s_UNselected.gif";
unselected[1] = new Image(); unselected[1].src =
"the80s_UNselected.gif"; unselected[2] = new Image();
unselected[2].src = "the90s_UNselected.gif";

var selectedTEXT = new Array(3); selectedTEXT[0] = new
Image();
selectedTEXT[0].src = "the70s_selected_TEXT.gif";
selectedTEXT[1] = new Image(); selectedTEXT[1].src =
"the80s_selected_TEXT.gif"; selectedTEXT[2] = new
Image(); selectedTEXT[2].src =
"the90s_selected_TEXT.gif";
```

Your two functions should look as follows:

```
function selectImage(imageNumber)
{
var selectStatement = "document." +
imageName[imageNumber]

+ ".src = selected[imageNumber].src;"; eval(
selectStatement );
document.textArea.src = selectedTEXT[imageNumber].src;
}

function deselectImage(imageNumber)
{
var deselectStatement = "document." +
imageName[imageNumber] + ".src = unselect eval(
deselectStatement );
document.textArea.src = "UNselected_TEXT.gif";
}
```

Your onmouseover and onmouseout event one-liners should look similar to the follows:

```
onmouseover="selectImage(0);" onmouseout="deselectImage(0)"
```

(where the 0 should be replaced by the appropriate image number);

## 15.8.10 Discussions of Activity 7

Your objects with Image properties should be set up in a way similar to the following:

```
var selected = new Object; selected.the70s = new Image();
selected.the70s.src = "the70s_selected.gif"; selected.the80s = new
Image(); selected.the80s.src = "the80s_selected.gif";
```

```

selected.the90s = new Image(); selected.the90s.src =
"the90s_selected.gif";

var unselected = new Object(); unselected.the70s = new Image();
unselected.the70s.src = "the70s_UNselected.gif"; unselected.the80s =
new Image(); unselected.the80s.src = "the80s_UNselected.gif";
unselected.the90s = new Image(); unselected.the90s.src =
"the90s_UNselected.gif";

var selectedTEXT = new Object(); selectedTEXT.the70s = new Image();
selectedTEXT.the70s.src = "the70s_selected_TEXT.gif";
selectedTEXT.the80s = new Image(); selectedTEXT.the80s.src =
"the80s_selected_TEXT.gif"; selectedTEXT.the90s = new Image();
selectedTEXT.the90s.src = "the90s_selected_TEXT.gif";

```

Your select and deselect functions can be simplified to the following:

```

function selectImage(imageName)
{
    document[imageName].src = selected[imageName].src;
    document.textArea.src = selectedTEXT[imageName].src;
}

function deselectImage(imageName)
{
    document[imageName].src = unselected[imageName].src;
    document.textArea.src = "UNselected_TEXT.gif";
}

```

See listing `music_associative_arrays.html` for a complete HTML file using the above code.

## 15.8.11 Discussions of Activity 8

Your constructor function should look something like the following:

```

function RolloverImage( nameString, selectedIMG,
unselectedIMG, selectedTEX
{
    this.name = nameString; this.selected = new
Image(); this.selected.src = selectedIMG;
this.unselected = new Image(); this.unselected.src
= unselectedIMG; this.selectedTEXT = new Image();
this.selectedTEXT.src = selectedTEXT;
this.unselectedTEXT = new Image();
this.unselectedTEXT.src = unselectedTEXT;
}

```

Your two instance methods, and their addition to the prototype should look similar to the following:

```

function selectImage()
{
    // access the IMG NAME by treating 'document' as an
associative array document[this.name].src =
this.selected.src;

textArea.src = this.selectedTEXT.src;
}

```

```

function deselectImage()
{
    // access the IMG NAME by treating 'document' as
    // an associative array
    document[this.name].src = this.unselected.src;
    textArea.src = this.unselectedTEXT.src;
}

RolloverImage.prototype.select = selectImage;
RolloverImage.prototype.deselect = deselectImage;

```

The rolloverImages object, and the three properties that are instances of RolloverImage should look similar to the following:

```

var rolloverImages = new Object();

rolloverImages.the70s = new RolloverImage( "the70s",
"the70s_selected.gif", "the70s_UNselected.gif",
"the70s_selected_TEXT.gif", "UNselected_TEXT.gif");

rolloverImages.the80s = new RolloverImage( "the80s",
"the80s_selected.gif", "the80s_UNselected.gif",
"the80s_selected_TEXT.gif", "UNselected_TEXT.gif");

rolloverImages.the90s = new RolloverImage("the90s",
"the90s_selected.gif", "the90s_UNselected.gif",
"the90s_selected_TEXT.gif", "UNselected_TEXT.gif");

```

Your HTML table should look as follows:

```

<TABLE CELLSPACING=0 CELLPADDING=0 WIDTH=202 BORDER=0>

<TR>
<TD rowspan = 3>
<IMG NAME="textArea" SRC="UNselected_TEXT.gif">
</TD>

<TD>
<A HREF="the70s.html" onMouseOver="rolloverImages.the70s.select()"
onMouseOut="rolloverImages.the70s.deselect()">
<IMG NAME="the70s" SRC="the70s_UNselected.gif" BORDER=0>
</A>
</TD>
</TR>

<TR>
<TD>
<A HREF="the80s.html" onMouseOver="rolloverImages.the80s.select()"
onMouseOut="rolloverImages.the80s.deselect()">
<IMG NAME="the80s" SRC="the80s_UNselected.gif" BORDER=0>
</A>
</TD>
</TR>

<TR>
<TD>
<A HREF="the90s.html" onMouseOver="rolloverImages.the90s.select()"
onMouseOut="rolloverImages.the90s.deselect()">
<IMG NAME="the90s" SRC="the90s_UNselected.gif" BORDER=0></A>

```

```
</TD>
</TR>

</TABLE>
```

## 15.8.12 Answers to Review Questions

1. An array is a tabular collection of data, accessed via a numeric index. Each element of an array can be a primitive value, or a reference to an object (possibly another Array object).
2. The number 4 is the index of the 5th element in the array (remember the first element is stored in array element `myArray[0]`). The index of an array states the element to which it is being referred.
3. There are 3 elements in this array. They are:

```
ages[0] ages[1] ages[2]
```

(remember, arrays are indexed with 0 as the first element index, not 1.)

4. Simple numeric variables like `a1` and `b1` store values, and when the value of variable `a1` is assigned to the value of `b1`, a copy of the value 5 is placed into variable `b1`. When `b1` changes the value it stores this has no effect on the number 5 stored elsewhere in memory in variable `a1`.

Array variables contain a reference to the location in memory where the array elements are stored. Copying an array variable means copying the reference, so in the above code variable `b2` is referring to the same array as `a2`, therefore changing the array elements referred to by `b2` will also change the elements referred to by `a2`.

5. In this statement `document` is a variable that contains a reference to the document instance of the Document constructor. When a browser loads and interprets an HTML file containing JavaScript, it creates a single instance based on Document, called `document`. This object is made available as a property of the global object to JavaScript programmers, so that they can refer to the properties and methods of browser `document`.
6. The answer is that `triangle1.width` has a value of 5. This is because when `t2` is assigned the value of `triangle1`, it is assigned the reference to `triangle1`. So when a change is made to the object referred to by `t2`, this is also the object referred to by `triangle1`.

Browser output is as follows:

---

```
triangle1 has width: 5
```

7. Yes they are. In fact more so that for many other object languages, since methods are just another form of data.

## 15.8.13 Contribution to Discussion Topics

1. Yes this statement is true — associative arrays and objects are the same thing. Looking at objects as associative arrays, however, provides a potentially more flexible and dynamic way to access properties and methods.

However, associative arrays should be distinguished from numerically indexed arrays, the latter having an explicit order to their elements.

2. When a prototype changes, the changes are inherited by all objects sharing the prototype at that point in time. Therefore if an object was previously created and used, and then its prototype changed, performing the same operations on the object will not necessarily result in the same behaviour. This could lead to inconsistent software system behaviour, and might be a very difficult bug to locate and repair.

Few circumstances exist where one might desire such changing behaviour from objects in the middle of

execution of a programme.

# Chapter 16. Interactive Images

## Table of Contents

Objectives .....	2
16.1 Introduction .....	2
16.1.1 Introduction to Image maps .....	2
16.1.2 The user's viewpoint of image maps .....	2
16.1.3 An example of user interaction with an image map .....	2
16.2 Server-side and client-side image maps .....	4
16.2.1 Server-side image maps .....	4
16.2.2 Client-side image maps .....	5
16.3 Implementing client-side image maps in HTML .....	5
16.3.1 Images .....	5
16.3.2 Relating a map and an image .....	6
16.3.3 Overview of HTML <MAP> tags .....	6
16.3.4 The <area> tag .....	6
16.3.5 Shape .....	7
16.3.6 Coordinates .....	7
16.3.7 Hyperlink .....	8
16.3.8 Title text .....	8
16.4 Reacting to mouse pointer location: onmouseover and onmouseout .....	9
16.4.1 Changing the contents of a text box .....	10
16.5 Summary exercise .....	11
16.6 Application and further work .....	11
16.6.1 Activity 4: Creating a rectangular area in an image map .....	11
16.6.2 Activity 5 — Creating a default hyperlink .....	11
16.6.3 Activity 6: Creating an image map for geometric shapes .....	12
16.6.4 Activity 7: Responding to onmouseover events .....	12
16.6.5 Activity 8: Controlling frame content from image maps .....	13
16.7 Review Questions .....	15
16.8 Frame hierarchies .....	15
16.9 Loading Multiple Documents .....	16
16.9.1 Loading two frames from one link .....	16
16.9.2 Loading more than two frames from one link .....	17
16.10 Answers .....	21
16.10.1 Discussions of Exercise 1 .....	21
16.10.2 Discussions of Exercise 2 .....	21
16.10.3 Discussions of Activity 1 .....	22
16.10.4 Discussions of Activity 2 .....	22
16.10.5 Discussions of Activity 3 .....	22
16.10.6 Discussions of Activity 4 .....	23
16.10.7 Discussions of Activity 5 .....	24
16.10.8 Discussions of Activity 6 .....	25
16.10.9 Discussions of Activity 7 .....	26
16.10.10 Discussions of Activity 8 .....	26
16.10.11 Discussions of Activity 9 .....	27
16.10.12 Discussions of the Review Questions .....	28

# Objectives

At the end of this unit you will be able to:

- Use image maps in HTML files;
- Implement client-side image maps;
- Implement mouse events.

## 16.1 Introduction

### 16.1.1 Introduction to Image maps

Image maps provide a way for users to interact with graphically presented information in a natural way. If you have spent time browsing websites with graphics you have probably navigated using image maps.

#### Definition

Image maps provide a mechanism for specified areas of GIF images to act as a hypertext link, and so link to other HTML pages or resources.

It is straightforward to create a hypertext reference for an entire image. For example:

```
<a href=index.html><img src=house.gif></a>
```

While code as the above is sufficient for images that act as buttons, there are times when navigation can be simplified, and made more intuitive and interesting by defining different areas of images to link to different hypertext anchors. Consider the following Web page navigation techniques:

- clicking on a city plan or country map to indicate where you live, and then being linked to a page showing details of the stores nearest to your area.
- using an information terminal in a shopping complex to click on the store you wish to visit, and then being taken to a page giving directions from the terminal location to the desired store.
- being presented with the shelves of a virtual, on-line grocery store (i.e. pictures of carrots, apples, bananas, grapes etc.), and clicking on the apples; and then being presented with pictures of different apple varieties, their quantities and available prices.
- being presented with a diagram of the product life cycle stages supported by an industry management consultancy firm, and clicking on the life cycle stage you are considering contracting them for. Then being presented with case studies, service/pricing examples and contact details.

The above scenarios require the browser/server to respond differently depending on which area of an image is clicked by the user. All these scenarios, and many more, can be implemented through the use of image maps.

### 16.1.2 The user's viewpoint of image maps

The user's experience of image maps is simple and intuitive: an image is shown on a Web page; the user clicks the part of the image that interests them, and a relevant hyperlink is followed.

### 16.1.3 An example of user interaction with an image map

Consider the following scenario:

A user wishes to purchase a spare part for their car.

## Advanced HTML

The user visits the website for a national chain of car maintenance stores.

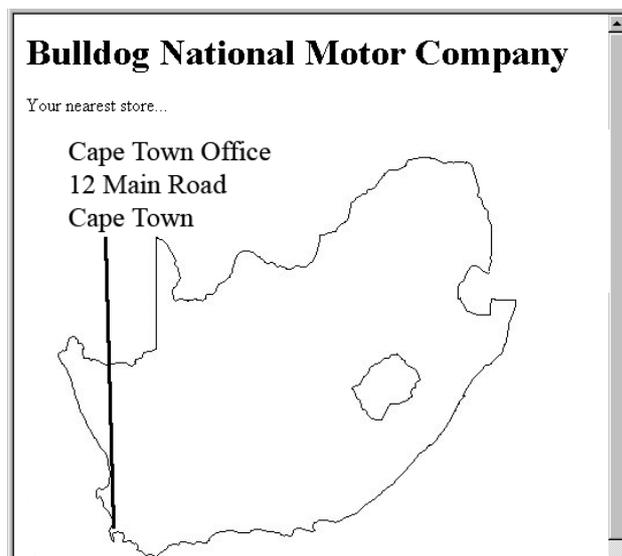
The user wishes to order and reserve the part in advance, and then pick up the part in person (to save delivery charges, and also to check that it fits their car while they are at the store) from their nearest store.

The user has a clear task to first locate the nearest store before both attempting to identify if the part is in stock and reserving/ordering it. This location task can be straightforward if the user can simply select the area they live in on a national map to be shown the location of the stores in the area.

In the figure below we see the user being presented with a map of South Africa and being invited to click where they live:



Let us assume that the user lives in Cape Town (the lower left part of this map of South Africa), and clicks there. The browser then responds by displaying another map, showing the address of the company's Cape Town office. See the diagram below:



If the user clicked Midrand, the browser would have responded by displaying the Johannesburg office. See the figure below:



## 16.2 Server-side and client-side image maps

The implementation of image maps requires the processing of the mouse click coordinates in conjunction with knowing the image areas and their associated hypertext links. The processing can take place in one of two places:

- On the server where the Web pages are stored.
- In the Web browser.

Originally, the only option for processing image maps was to do so on the server. However, later versions of HTML have added features allowing client-side processing. Many people are now developing their new pages (and updating older pages) to process image maps in the Web browser itself, thus reducing the amount of processing required by the server.

### 16.2.1 Server-side image maps

Server-side image mapping uses images, .map files and CGI scripts. Often server-side image maps use the Imagemap programme.

The programme processing image maps needs to be on the server, usually in the cgi-bin/ directory.

The HREF attribute of server side imagemaps is always a hyperlink to the cgi-bin directory:

```
<a href="/cgi-bin/imagemap/user_dir/image.map">

</a>
```

In fact, the HREF path can sometimes be interpreted as a combination of two paths:

- The location of the "imagemap" programme, such as /cgi-bin/imagemap
- The location of the (coordinate) map for the image, user\_dir/image.map. This describes the various areas of the image.

A major disadvantage of server-side image maps is that their implementation can differ between different server systems. This means that the author of the website using image maps needs to communicate closely with the Web server's administrator to ensure that the maps work correctly.

Since there are so many advantages to client-side imagemaps, and almost all current browser versions support them, we shall not explore server-side image maps further in this unit (although most image map programmes still provide

features to output/edit server-side map files).

## 16.2.2 Client-side image maps

Clients-side image maps have a number of advantages over server-side maps.

- Reduced server load, since the mapping is done on the client.
- There is no need to communicate with server side scripts.
- The user can see the image maps associated hypertext URLs in the status bar of their browser before clicking.

## 16.3 Implementing client-side image maps in HTML

Two things are need in order to implement client-side image maps:

- An image.
- A map of the image areas and associated hyperlinks.

### 16.3.1 Images

Most images created for Web pages are in either in JPEG or PNG format. The first step to developing an image map is to ensure that you have a version of your image in a format supported by imagemaps for the browsers you are using. The old GIF image format is widely supported by image maps, but all modern browsers also support newer formats, such as PNG.

If the image is not already in the format you wish to use (such as PNG), you will need to convert it. The GIMP [www.gimp.org] is a Free Software image editor, similar to Adobe's Photoshop, that will allow you to not only convert images to the format you require, but to also edit the images. The software is free for both commercial and non-commercial use.

#### Preparation activities

You may find it useful to perform the following activities in preparation for the next sections:

#### Activity 1: Creating a PNG image

Mr. Green's Grocery Company wishes to start trading on-line.

Locate or create a picture of three or four different vegetables in a single PNG image (e.g. an area of carrots, apples and bananas). This PNG file will be needed for some of the later activities.

Store this image in a directory called "on\_line\_grocer".

You can find a discussion of this activity at the end of the unit.

#### Activity 2: An HTML file to display an image

Create an HTML file called food.html that displays your vegetables image. Use the heading "Mr Green's On-line Groceries", and display the text "Click on the food you wish to order."

You can find a discussion of this activity at the end of the unit.

#### Activity 3: Creating an "out of stock" HTML page

Create an HTML file called "out\_of\_stock.html".

Make this a simple page that has the same "Mr Green's On-Line Grocery" heading as previously, and then the message: "Sorry we are presently out of stock of your selected food." (this page is required as part of a later activity)

You can find a discussion of this activity at the end of the unit.

## 16.3.2 Relating a map and an image

The HTML IMG tag is used to display the image:

```
<IMG src="foods.gif" >
```

To associate an image with a particular imagemap, more attributes are provided to the IMG tag. This code associates the foods.gif image with an imagemap named fruit\_and\_veg:

```
<IMG src="foods.gif" usemap="#fruit_and_veg">
```

The border is often set to zero (i.e. so there is no outline), and an alternative text message for the image is defined as well:

```
<IMG src="foods.gif" alt="foods for purchase" usemap="#fruit_and_veg"
border="0"
```

### Note

Note that the hash '#' symbol precedes the image map filename in the IMG tag.

## 16.3.3 Overview of HTML <MAP> tags

An imagemap needs to be associated with the image. A <map> is an HTML tag defining the active image areas and corresponding hypertext links (and other attributes of the areas).

The generic form of a map is as follows:

```
<map name="MAPNAME" >
  <area shape="SHAPE" coords="X1, Y1, ..." href="HYPERLINK" ...>
  <area ... >
  <area ... >
</map >
```

where:

- "MAPNAME" is the map's chosen name (to be specified in the IMG tag that displays your image).
- "SHAPE" is one of "rect", "circle" and "poly" — specifying the shape being defined.
- "X1, Y1, ..." is the set of coordinates defining an image area.
- "HYPERLINK" is a standard hyperlink, and is the hyperlink that is followed when the user clicks inside the defined image area.

An example of a map is:

```
<map name="geometry" >
  <area shape="rect" coords="10,10,50,50" href="square.html" >
  <area shape="circle" coords="100,100,40" href="circle.html" >
  <area shape="poly" coords="100,200,200,200,200,300,100,200"
href="triangle
</map >
```

## 16.3.4 The <area> tag

As you can see from the previous section, a map consists of a set of areas. Each area can have following attributes defined:

- the shape of the area being defined.
- the coordinates (cord) that define the boundaries of the area.
- the hyperlink (href) to be followed when the user clicks inside the area.
- title text (title) to be displayed next to the mouse pointer when over the area.

The Javascript actions to execute when the mouse pointer moves over the area (onMouseOver) or moves away from the area (onMouseOut)

We shall consider each of these in turn.

## 16.3.5 Shape

Consider the example map again:

```
<map name="geometry" >
<area shape="rect" cords="10,10,50,50" href="square.html" >
<area shape="circle" cords="100,100,40" href="circle.html" >
<area shape="poly" cords="100,200,200,200,200,300,100,200"
href="triangle.html" >
</map >
```

This example illustrates the three different shapes possible with the area tag:

- rectangles *rect*
- circles *circle*
- polygons *poly*

### Note

Some browsers accept other versions of these shape names, such as circ, rectangle, polygon.

Any other kind of shape can be approximated by creating a polygon with many edges.

The shape is defined by providing the shape name in quotation marks. For example, to define a circular area one would write the following:

```
<area shape="circle" ... >
```

## 16.3.6 Coordinates

Each shape type requires a different set of coordinates to define the area boundaries.

The coordinates are provided in quotation marks, and separated by commas. For example the values to define a rectangle might be:

```
cords="10,10,50,50"
```

**Exercise 1: coordinates required for defining area shapes**

Notice in the example how the number of coordinates is different for each shape:

- 4 coordinates for the "rect"
- 3 coordinates for the "circle"
- 8 coordinates for the "poly"

Why is this? What do you think the three coordinates for the circle define? You can find the answers to this exercise at the end of the unit.

## 16.3.7 Hyperlink

The hyperlink to be followed when the user clicks within an area is defined just as any other hyperlink. For example a link to a page "products.html" would be defined as follows:

```
href="products.html"
```

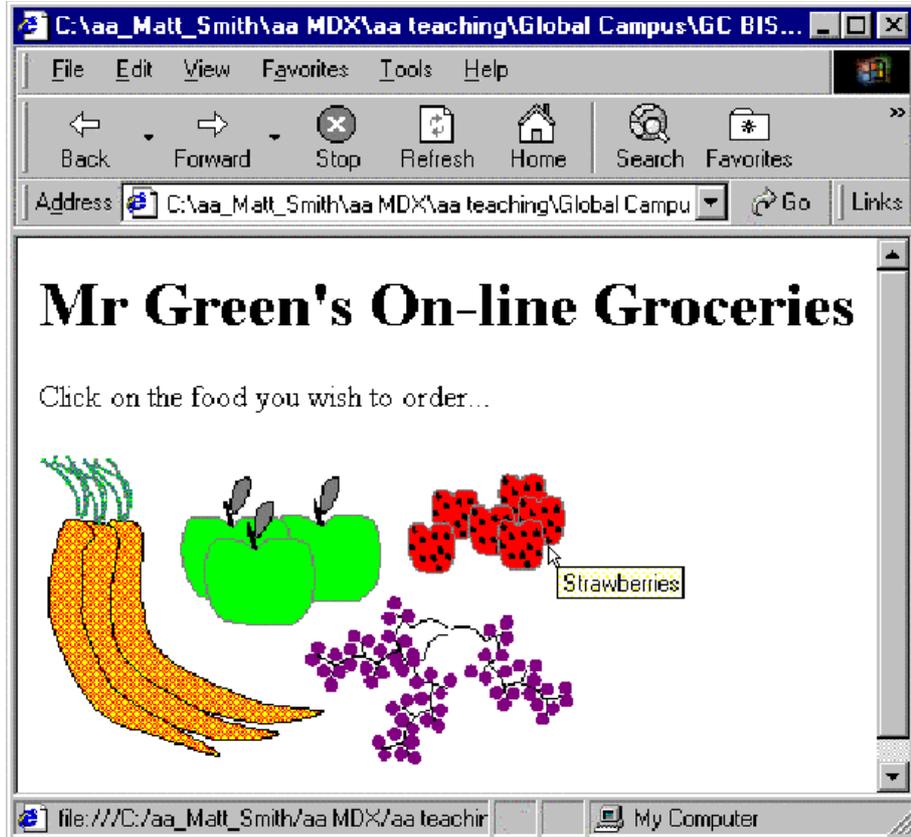
## 16.3.8 Title text

Text can be provided that is displayed when the mouse is over the area. This can give the user an idea of the area's boundaries, and also what might occur if the area is clicked.

This text is defined with the "title" attribute. For example to define title text of "Strawberries" one would write:

```
title=" Strawberries"
```

The figure below illustrates title text being displayed by the user's mouse pointer when over an area:



## 16.4 Reacting to mouse pointer location: onmouseover and onmouseout

It is possible to execute JavaScript commands when the mouse pointer moves over the area, and also when the mouse pointer moves from the area to another part of the image.

An image changing when a mouse pointer moves over or away from it is called image rollover;

This feature can be used to create a number of different interaction effects. Examples of the kinds of actions performed when these events are detected include;

- the status bar text can display information about the area the pointer is over;
- some other part of a page can be changed to display information (e.g. the contents of a text box);
- in a framed system, the page displayed in the frame can be changed without the user having to click;
- in a non-framed system the page (or image) displayed can be changed as the mouse moves over different areas.

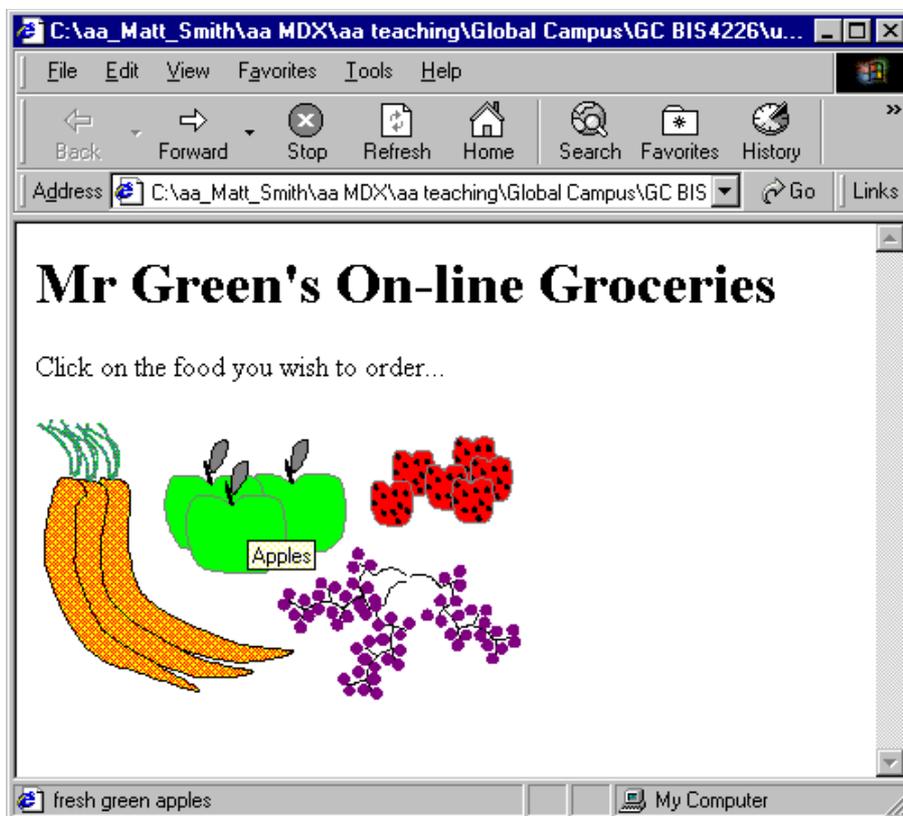
Setting the status bar

To set the status bar to display 'fresh green apples'. one could use the line:

```
<area shape="rect" coords="10,10,50,50" title="Apples"
onmouseover="window.status='fresh green apples'; return true">
```

Remember to 'return true' when setting the status bar text.

The figure below illustrates the browser window when the mouse pointer is over the apples area:



To clear the status bar when the mouse pointer moves away from the apples area one could use the line:

## Advanced HTML

```
<area shape="rect" coords="10,10,50,50" title="Apples"
onMouseOver="window.status='fresh green apples'; return
true"
```

```
onMouseOut="window.status='' ; return true">
```

Again, remember to 'return true' when setting the status bar text.

### 16.4.1 Changing the contents of a text box

If we extend the on-line grocery example by adding a text box at the bottom of the page, we can set the text in this box using onmouseover and onmouseout events.

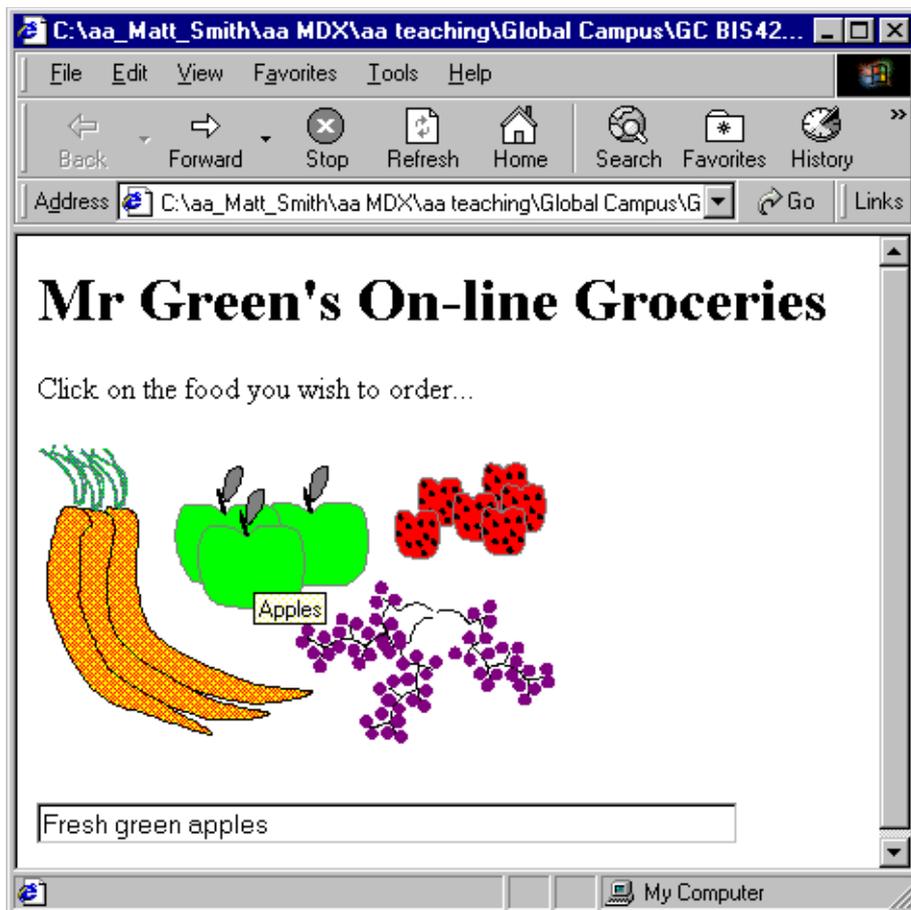
We add the text box by adding code for a very simple form:

```
<form name=myform>
<input type=text name=food_details size=50>
</form>
```

We can now add the following onmouseover and onmouseout actions:

```
onmouseover="document.myform.food.value='Fresh green
apples'" onmouseout="document.myform.food.value=''"
```

The screen looks as follows when the mouse is over the apples area:



## 16.5 Summary exercise

### Exercise 2: Summary of Image Maps

Complete the following statements that summarize the concepts explored in this unit: Image maps allow a Web page designer to make \_\_\_\_\_ of an \_\_\_\_\_ to behave differently. Each defined area can be \_\_\_\_\_, circular or a many-sided shape defined by a \_\_\_\_\_. A \_\_\_\_\_ area is defined with 4 coordinates.

A \_\_\_\_\_ area is defined with 3 coordinates.

A \_\_\_\_\_ area is defined with an even number of values, representing coordinates for each point defining its boundaries.

The \_\_\_\_\_ attribute of <area...> tags can be used to make a piece of text appear next to the mouse pointer when it is over a defined area.

An area can be made to execute a line of \_\_\_\_\_ in response to \_\_\_\_\_ and onMouseOut event.

In the early days image maps were implemented using \_\_\_\_\_ techniques. These days most people write client-side image maps.

An image changing when a mouse pointer moves over or away from it is called image \_\_\_\_\_.

You can find the answers to this exercise at the end of the unit.

## 16.6 Application and further work

### 16.6.1 Activity 4: Creating a rectangular area in an image map

Inside your food.html file create an image map called "fruit\_and\_veg".

In your "fruit\_and\_veg" image map, create a rectangular area around one of your groups of vegetables (for example in the "foods.gif" image the red strawberries seem to lend themselves to a rectangular area.

#### Hint

You can either estimate the coordinates if you know the image size, or use freely available software such as the GIMP (previously mentioned).

Make your defined area hyperlink to your HTML page "out\_of\_stock.html".

#### Hint

Remember to extend your IMG tag to state that the "fruit\_and\_veg" map should be used.

You can find a discussion of this activity at the end of the unit.

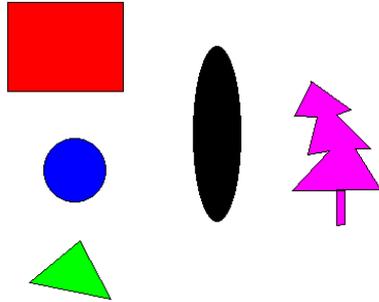
### 16.6.2 Activity 5 — Creating a default hyperlink

Edit your map so that title text is displayed when the mouse pointer is over your area. So, for example, for our foods.gif image and the rectangle around the strawberries the title text chosen is "Strawberries".

You can find a discussion of this activity at the end of the unit.

### 16.6.3 Activity 6: Creating an image map for geometric shapes

Create an image map for the following image (shapes.gif):



Set appropriate title text for each area you define: so the title text for the red rectangle should be 'Red Rectangle', and for the blue circle should be 'Blue Circle' and so on.

#### Hint

If you do not have the use of an image map tool, the top left of the image is, as always, (0, 0); the bottom right will be the height and width of the image.

You can find a discussion of this activity at the end of the unit.

### 16.6.4 Activity 7: Responding to onMouseOver events

Consider the three pages below. The pages have been created using with the three GIF images: home.gif, products.gif and contact.gif.





Create three HTML files (index.html, products.html and contact.html) that look as above. Note that the home.gif image should be displayed in the index.html file, since index.html is a convention to follow for the home page of a site.

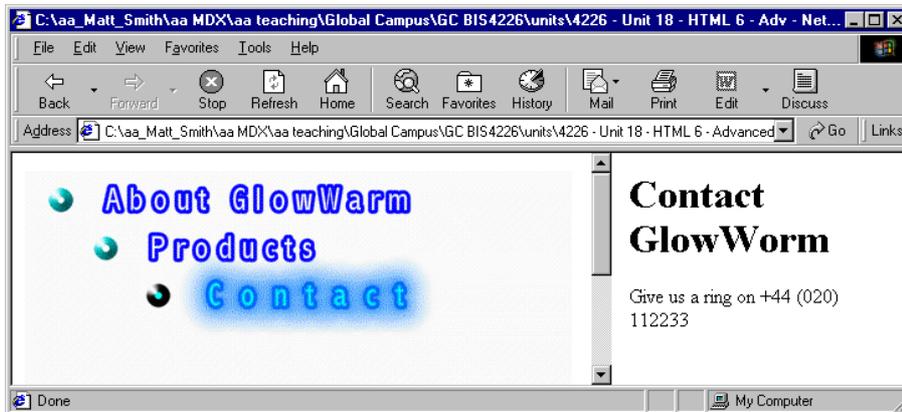
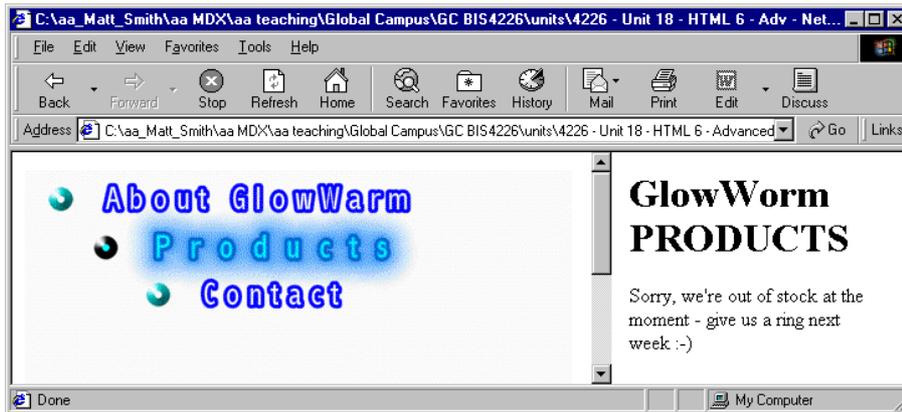
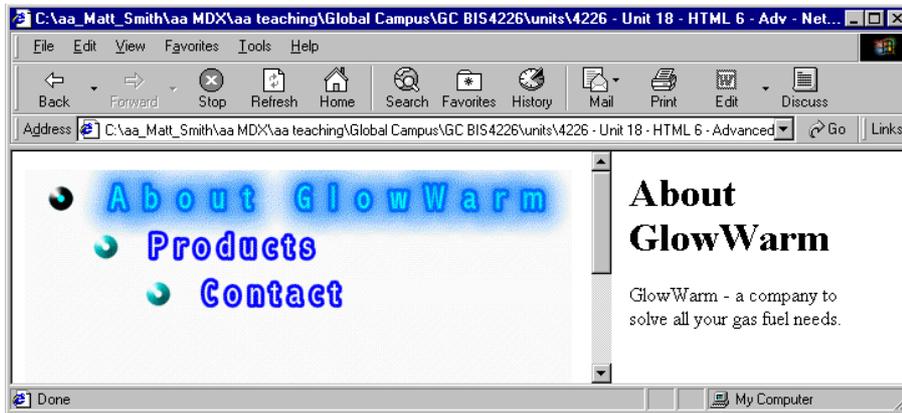
Create an image map for each image, so that as the mouse pointer passes over the word "Products", the "products.html" file is displayed etc. Thus the user should be able to navigate without having to click the mouse at all.

You can find a discussion of this activity at the end of the unit.

### 16.6.5 Activity 8: Controlling frame content from image maps

Consider the three pages below. The pages have been created using with the three GIF images: about\_nav.gif, products\_nav.gif and contact\_nav.gif.

## Advanced HTML



These pages illustrate how the contents of the right hand frame can be controlled from a simple Javascript command coded in an image map onMouseOver attribute.

The .html pages displayed in the right hand frames are: about.html, products.html and contact.html.

Create three HTML files to display the GIFs and define the image maps. The image maps use the onMouseOver attributes to load the appropriate files into the right and left frames.

You will also need an initial file (index.html) to set up the frames and load the initial files into each frame.

Call your HTML files:

- index.html (to display the about\_nav.html and about.html files)
- about\_nav.html
- products\_nav.html
- contact\_nav.html

You can find a discussion of this activity at the end of the unit.

## 16.7 Review Questions

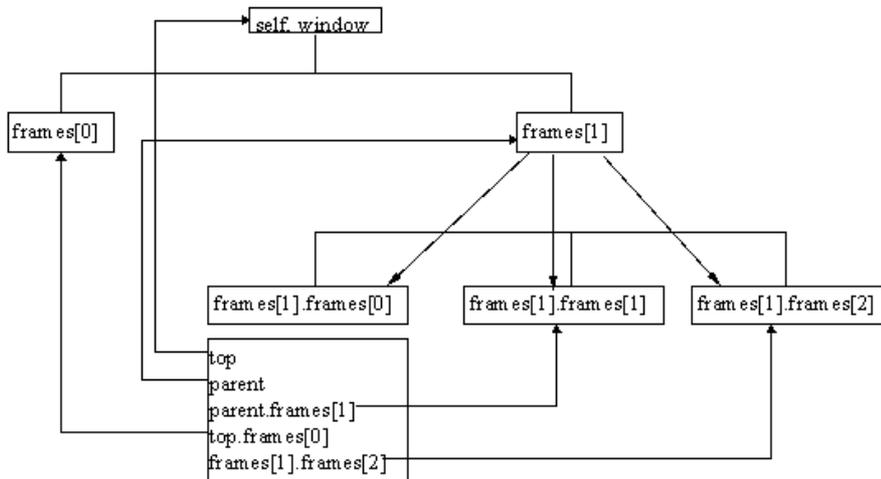
1. What are the different image map shapes that can be defined?
2. Which are more 'portable': server-side or client-side image maps?
3. Discuss this statement: You should keep things consistent and simple by always defining all areas as polygons.
4. Discuss this statement: You should always define contiguous (physically touching) areas for all parts of an image, so wherever the user clicks, the nearest relevant hyperlink is followed.

You can find the discussions on these review questions at the end of the unit.

## 16.8 Frame hierarchies

When you are creating a complicated frameset, some planning should be used to avoid the replication of Web pages inside Web pages (as highlighted in the example in the disadvantages section). This section is dedicated to frame hierarchies and constructing a hierarchy to avoid the problem self replication.

Below is an example frameset hierarchy. Note that the JavaScript included at the bottom level indicates the commands necessary to load an HTML document into the respective.html frame.

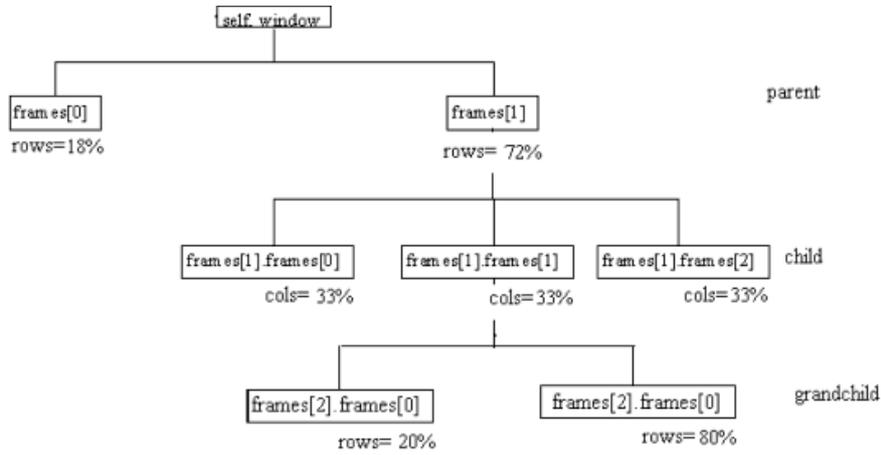


### ACTIVITY 9

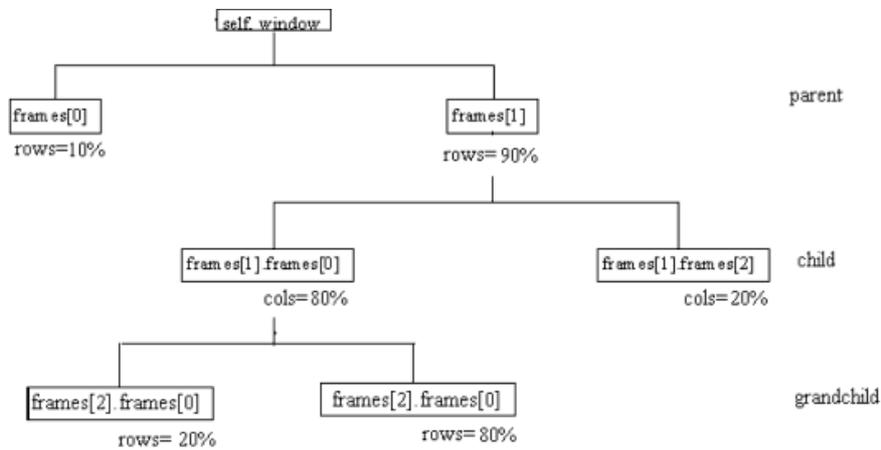
In this Activity you will design Web pages using given frame hierarchies.

1. Design the frameset for the hierarchy shown below:

## Advanced HTML



2. Design the frameset for the hierarchy shown below:



You can find a discussion of this activity at the end of the unit.

## 16.9 Loading Multiple Documents

### 16.9.1 Loading two frames from one link

This section will look at how to load two documents with one click of a button. Take a look at Frames Example No 5, which will once again load in a new browser. In this example, the contents of both frames change with the click of one button.

This occurs when the user interacts with the user interface. In order to explain how to do this, some terminology has to be introduced and defined in the table below:

Term	Definition	Example
Object	An object is an entity in the real world, or, as in our case, an entity on a Web page. It has both state and behaviour.	link, button, image, frame, from, window, text

## Advanced HTML

Event	An event causes the behaviour of an object to change, usually generated by the user interacting with the graphical interface.	mouse click, mouse move
Event handler	When an event occurs an associated function is invoked to handle the objects behaviour. The event handler usually has a name closely associated with the event.	onClick, onLoad, onUnload, onSubmit etc...

There are many event handlers. In this case when the event "mouse click" occurs on the specific "link" object it should be managed by the "onClick" event handler to display two HTML documents simultaneously in their respective frames.

### 16.9.2 Loading more than two frames from one link

A more complex example involves changing the contents of more than one frame. In this example, clicking a link (metals, fruits or animals) in frame 1 changes the contents of frames 2, 3 and 4, and clicking the contents link in the index frame re-sets the contents of all the frames. (Note: you will build this example in Activity 10 below.)

#### Activity 10: Loading multiple documents with one click

Load your HTML editor, e.g. MS Notepad, and create the Web page that sets up the five frames in a nested frameset, and then save the file as act3.html. The code is shown below.

```
<html>
<head>
<title>Activity 10</title>
</head>
<frameset cols="150,*">
<frame src="main.html" name="index" >
<frameset rows="20%,*">
<frame src="frame1.html" name="frame1">
<frameset cols="*,*,*">
<frame src="frame2.html" name="frame2">
<frame src="frame3.html" name="frame3">
<frame src="frame4.html" name="frame4">
</frameset>
</frameset>
</frameset>
</html>
```

1. Create the five files to appear in the five frames, ie.main.html, frame1.html, frame2.html, frame3.html and frame4.html:

main.html is the page that appears in the index frame.

```
<html>
<head>
</head>
<body bgcolor="darkkhaki">
<h2>index frame</h2>
    The colour of this frame is
<br>
<center><i>dark khaki</i></center>
<p>
<a href="act3.html">contents</a>
```

## Advanced HTML

```
</body>
</html>
```

frame1.html holds the links that will load the metals, fruits and animals pages into frames 1, 2, 3 and 4. Note the JavaScript required to do this.

```
<html>
<head>
<script language="javascript">
function LoadInChildFrames(pageForFrame2,
pageForFrame3,pageForFrame4
{parent.frame2.location.href=pageForFrame2
parent.frame3.location.href=pageForFrame3
parent.frame4.location.href=pageForFrame4}
</script>
</head>
<body bgcolor="gold">
<h2>frame 1</h2>
<a href="metals.html"
onclick="LoadInChildFrames('gold.html', 'silver
<a href="fruits.html"
onclick="LoadInChildFrames('orange.html', 'lemon
<a href="animals.html"
onclick="LoadInChildFrames('lion.html', 'elephant
</body>
</html>
```

frame2.html, frame3.html and frame4.html are the initial pages in the lower frames. frame2.html

```
<html>
<head>
</head>
<body bgcolor="mediumspringgreen">
<h2>frame 2</h2>
The colour of this frame is <br>
<center><i> medium spring green</i></center>
</body>
</html>
```

frame3.html

```
<html>
<head>
</head>
<body bgcolor="cornflowerblue">
<h2>frame 3</h2>
The colour of this frame is <br>
<center><i> corn flower blue</i></center>
</body>
</html>
```

frame4.html

```
<html>
```

## Advanced HTML

```
<head>
</head>
<body bgcolor="blanchedalmond">
<h2>frame 4</h2>
The colour of this frame is <br>
<center><i> blanché almond</i></center>
</body>
</html>
```

2. Now create the files for the respective links (metals, fruits and animals). Note, that only the.html code within the <body> is shown here.

### metals.html

```
<body bgcolor="slategray" text="yellow">
<h2>frame 1</h2>
welcome to the metals page, where you can find information on
many of the
</body>
```

### gold.html

```
<body bgcolor="gold">
<h2>frame 2</h2>
welcome to the gold page
</body>
```

### silver.html

```
<body bgcolor="silver">
<h2>frame 3</h2>
welcome to the silver frame
</body>
```

### bronze.html

```
<body bgcolor="tan">
<h2>frame 4</h2>
welcome to the bronze frame
</body>
```

### fruits.html

```
<body bgcolor="slategray" text="yellow">
<h2>frame 1</h2>
welcome to the fruits page, where you can find information on
many of the
</body>
```

### orange.html

## Advanced HTML

```
<body bgcolor="orange">
<h2>frame 2</h2>
welcome to the orange page
</body>
```

### lemon.html

```
<body bgcolor="yellow">
<h2>frame 3</h2>
welcome to the lemon page
</body>
```

### lime.html

```
<body bgcolor="lime">
<h2>frame 4</h2>
welcome to the lime page
</body>
```

### animals.html

```
<body bgcolor="slategray" text="yellow">
<h2>frame 1</h2>
welcome to the animals page, where you can find information on
many of th
</body>
```

### lion.html

```
<body bgcolor="goldenrod">
<h2>frame 2</h2>
welcome to the lion page
</body>
```

### elephant.html

```
<body bgcolor="ivory">
<h2>frame 3</h2>
welcome to the elephant page
</body>
```

### camel.html

```
<body bgcolor="palegoldenrod">
<h2>frame 3</h2>
welcome to the camel page
</body>
```

3. Load act3.html and test the metals, links and fruits links in frame 1. You may notice that if you click

## Advanced HTML

on the contents link in the index frame, the entire page loads itself inside the frame. This problem can be resolved by adding JavaScript to ensure that the contents page (main.html) loads in the top frame. The script checks if the window containing the link is the top window. If it is not, then the top location is changed. Add the following JavaScript in the <head> section of act3.html

```
<script language="javascript"> if(top!=self)
    top.location.href = location.href
</script>
```

4. Finally, test the file again in your browser.

# 16.10 Answers

## 16.10.1 Discussions of Exercise 1

Different shapes are defined in different ways.

Remember the origin of an image on a Web page is the top left, not bottom left as you may be used to with graphs on paper and in mathematics classes.

A RECTANGLE can be defined in many different ways, the most common two methods are to state the X and Y coordinates of the corner closest to the origin (0,0), and then to either state the coordinates of the opposite corner, or to state the width and height of the rectangle. The developers of HTML decided to define rectangles with four coordinates, where the first two are the X and Y values for the top left corner, and then the X and Y coordinates for the bottom right corner.

A CIRCLE can be easily defined using three values, the X and Y coordinates of the centre of the circle, and the radius of the circle.

A POLYGON is a shape whose edges are defined by a set of straight lines: squares, rectangles, triangles, hexagons are all examples of polygons. Since different polygons can have different numbers of sides, polygon areas for maps are defined simply by a sequence of pairs of (X, Y) coordinates for the end points of each straight line.

It is possible to use polygons to define rectangles, and even to approximate circles: some people just use polygons for all areas, although for rectangles and circles this results in more points being defined than necessary.

## 16.10.2 Discussions of Exercise 2

Suggested completed summary statements are as follows:

Image maps allow a Web page designer to make areas of an image to behave differently. Each defined area can be rectangular, circular or a many-sided shape defined by a polygon. A rect area is defined with 4 coordinates.

A circle area is defined with 3 coordinates.

A poly area is defined with an even number of values, representing coordinates for each point defining its boundaries.

The title attribute of <area...> tags can be used to make a piece of text appear next to the mouse pointer when it is over a defined area.

An area can be made to execute a line of Javascript in response to onMouseOver and onMouseOut event.

In the early days image maps were implemented using server-side techniques. These days most people write client-side image maps.

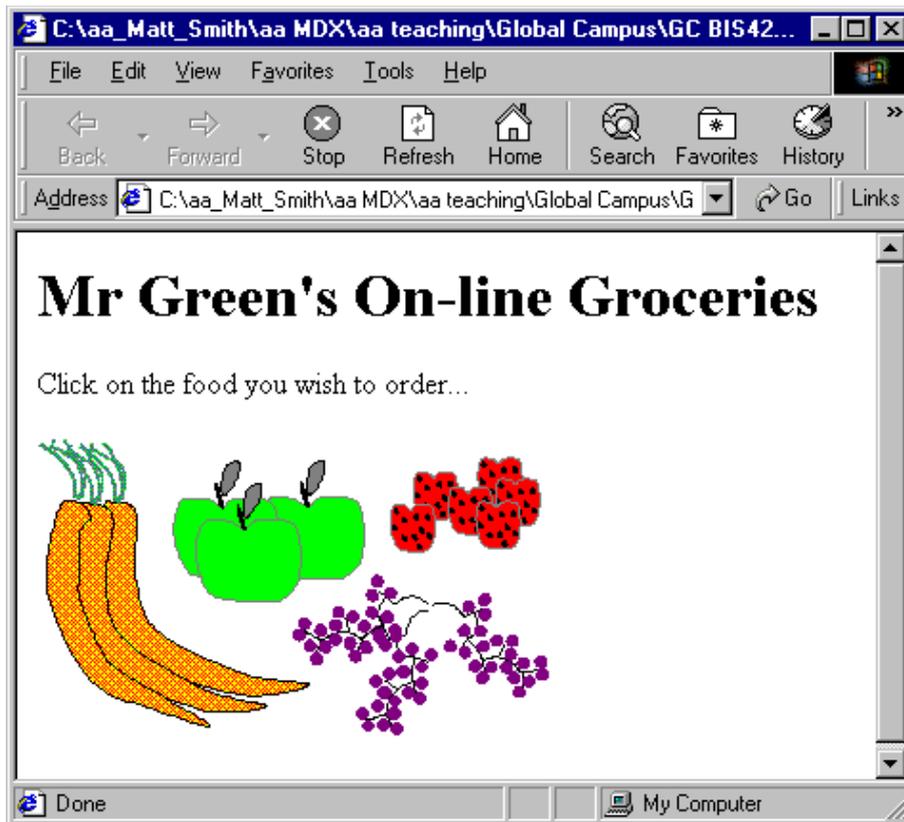
An image changing when a mouse pointer moves over or away from it is called image rollover.

### 16.10.3 Discussions of Activity 1

You will have created or located a picture (or perhaps a photograph) of different vegetables. This file should be copied to a directory you have created called "on\_line\_grocer".

### 16.10.4 Discussions of Activity 2

You will have created an HTML file that when displayed looks something like the following:



The source of your HTML file will look something like the following (although you will more than likely be using a PNG image rather than a GIF):

```
<HTML>
<BODY>
<H1>Mr Green's On-line Groceries</H1>
<p>Click on the food you wish to order...
<p>
<IMG src="foods.gif" >
</BODY>
</HTML>
```

### 16.10.5 Discussions of Activity 3

You will have created an HTML file that, when displayed, looks something like the following:



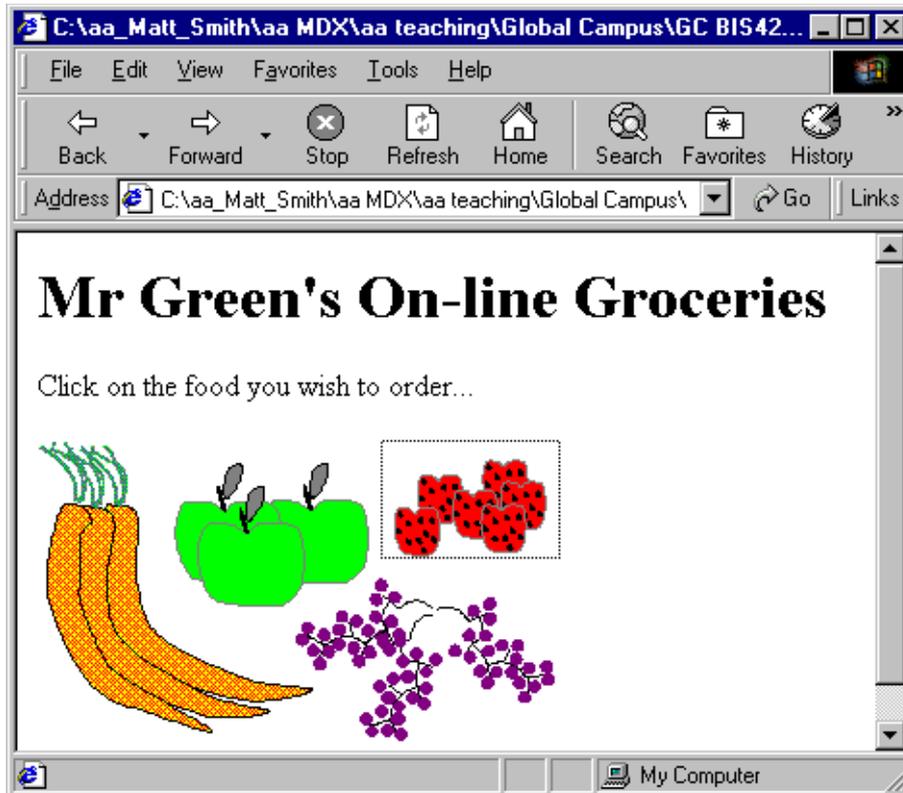
The source of your HTML file will look something like the following:

```
<HTML>
<BODY>
<H1>Mr Green's On-line Groceries</H1>
  Sorry we are presently out of stock of your selected food
</BODY>
</HTML>
```

## 16.10.6 Discussions of Activity 4

You will have amended your HTML file so that, while it does not appear any different when displayed, a rectangular region is now linked to your "out\_of\_stock.html" file.

The rectangle created around the strawberries for the foods.gif image is illustrated below:



Your HTML code should look similar to the following (this complete.html file can be found in the file food2.html):

```
<HTML>

<BODY>

<H1>Mr Green's On-line Groceries</H1> Click on the food you wish to
order...

<p>

<IMG src="foods.gif" alt="foods for purchase"
usemap="#fruit_and_veg" border="0

<map name="fruit_and_veg">

    <area shape="rect" coords="178,0,271,62"
href="out_of_stock.html ">

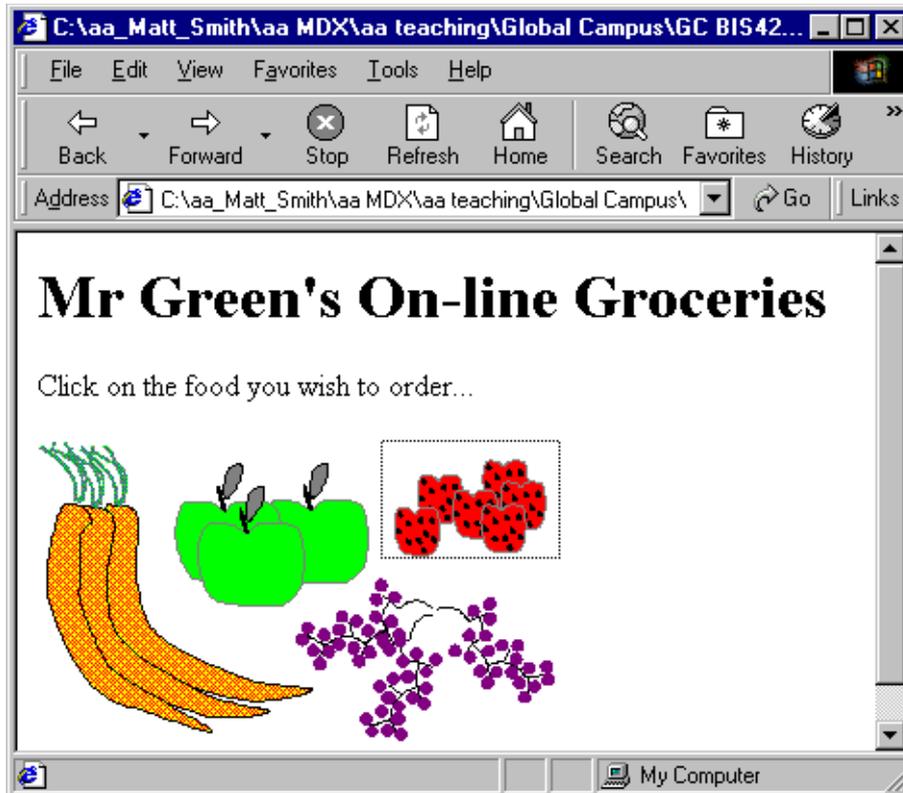
</map>

</BODY>

</HTML>
```

## 16.10.7 Discussions of Activity 5

When the mouse is over your rectangular area the title text should be displayed by the mouse pointer. For example for our image the screen looks as follows:



The only change to the code of the map is to add the "title" attribute to the <area ... /> line. For our example we changed the line from:

```
<area shape="rect" coords="178,0,271,62"
href="out_of_stock.html">
```

to

```
<area shape="rect" title="Strawberries"
coords="178,0,271,62" href="out
```

## 16.10.8 Discussions of Activity 6

The rectangle and circle shapes should be easy with the 'rect' and 'circle' shapes. The green triangle and purple tree can be achieved with the 'poly' shape.

The black oval is more difficult, and would usually either be implemented by a bounding rectangle, or approximated with a 'poly' polygon of 4-8 sides.

The listing for an imagemap for these shapes (and also defining the title text) is as follows:

```
<map name="shapes">
<area shape="rect" coords="47,42,159,129" nohref title="Red Rectangle">
<area shape="circle" coords="112,205,32" nohref title="Blue Circle">
<area shape="poly" coords="117,275,147,332,69,316,116,276,116,276" nohref
title
<area shape="poly"
coords="273,167,269,224,255,256,248,257,229,214,226,155,235,
```

```

<area shape="poly"
coords="374,259,375,226,409,226,385,185,398,185,368,150,379,

</map>

```

As can be seen, for shapes like the oval and tree, many sided polygons are needed, and effective image map creation requires the use of tools, since hand calculation and coding of such lists of coordinates is very time consuming.

## 16.10.9 Discussions of Activity 7

The.html files simply have to display the heading "GlowWarm" products and the appropriate image. The index.html file, including image map, should look something like the following:

```

<html>

<h1>GlowWarm Products</h1>

<img src=home.gif usemap="#home" border="0">

<map name="home">

<area shape="rect" coords="47,41,249,79" href="products.html"
title="Link to pr onMouseOver="parent.location.href='products.html'
">

<area shape="rect" coords="86,80,272,118" href="contact.html"
title="Link to co onMouseOver="parent.location.href='contact.html'
">

<area shape="default" nohref>

</map>

```

For each file, there should be two active areas defined in the image map — for the two other files. So in the index.html file there are image map areas defined to link to products.html and contact.html. For the products.html file there are links to index.html and contact.html, and so on.

## 16.10.10 Discussions of Activity 8

The index.html file needs to load the about\_nav.html file into the left hand frame and the about.html file into the right hand frame:

```

<html>

<frameset cols="480,*" border=1>

<frame src="about_nav.html" name="navbar">

<frame src="about.html" name="content">

</frameset>

</html>

```

The about\_nav.html file simple needs to display the about\_nav.gif image, and set up the image map:

```

<html>



<map name="about_nav">

<area shape="rect" coords="38,43,249,79" nohref title=""
onMouseOver="parent.fr

<area shape="rect" coords="83,80,264,118" nohref title=""
onMouseOver="parent.f

</map>

</html>

```

As you can see in the code above, there are two areas defined, one for products and one for contact.

For each area there are two Javascript lines for the onMouseOver attribute. For example when the mouse pointer goes over the products area:

```

onMouseOver="

parent.frames[1].location.href='products.html' ;

parent.frames[0].location.href ">

```

First the products.html file is loaded into the right hand frame (frames[1]) then the products\_nav.html file is loaded into the left hand frame (frames[0]). The semi-colon allows more than one Javascript statement to be defined for an onMouseOver or onMouseOut attribute.

## 16.10.11 Discussions of Activity 9

1.

```

<FRAMESET ROWS = "18%,72%">
<FRAME src="a.html">
<FRAMESET COLS="33%,33%,33%">
<FRAME src="b.html">
<FRAMESET ROWS="20%,80%">
<FRAME src=c.html>
<FRAME src=d.html>
</FRAMESET>
<FRAME src=e.html>
</FRAMESET>
</FRAMESET>
</HTML>

```

2.

```

<FRAMESET ROWS = "10%,90%">
<FRAME src="a.html">
<FRAMESET COLS="80%,20%">
<FRAMESET ROWS="20%,80%">
<FRAME src=b.html>
<FRAME src=c.html>
</FRAMESET>
<FRAME src=d.html>

```

```
</FRAMESET>  
</FRAMESET>  
</HTML>
```

## 16.10.12 Discussions of the Review Questions

1. There are three kinds of shape: rectangular (rect), circular (circle) and polygon (poly).

Any shape that is not rectangular or circular can be approximated with a many-sided polygon.

2. Client-side image maps are more portable, since all that is required is the image and the Web page that displays the image and contains the image map. Server side image maps require CGI programmes like `imagemap`, and the map stays located on the server.

Client-side image maps can be stored on non-CGI devices, like a set of Web pages on a static CD-ROM for example.

3. We shall consider the answer separately for rectangular areas (rect) and for circular areas (circle).

It is possible to define rectangular areas with polygons. This takes a little more code in the `.html` file (since four points need to be defined, not just two). But there is significantly more processing involved in deciding if the mouse is inside a polygon, compared to whether the mouse cursor is inside a rectangle.

Circular areas can be approximated with polygons (in fact in any digital imaging system circles must always be approximated). A circle can be defined with three values using `circle`, but to approximate a curve with a polygon can take 10s or even 100s of lines, depending on the size of the circle. Also, the algorithm to determine if a mouse pointer is within a circular area can sometimes be written simpler than the one to determine if a cursor is inside a polygon (depending on the number of sides in the polygon).

So, there are both HTML file size, and algorithmic processing efficiency reasons, for using `rect` and `circle` wherever possible.

4. There are times when such an approach is useful. For example recall the Bulldog South Africa map, where the user does desire the location of the nearest store to where they click.

However, for images made up of several visual components, it can be frustrating to users to clearly click away from one visual object, but to be taken to that object's hyperlink. Better in these cases to define areas around each visual object, and leave the other parts of the image inactive.

Many Web designers give some leeway, so where one object in an image is not right next to another, the area defined is a little out from the visual object itself — if using rectangles or circles to approximate the shapes of objects, you should err on the side of caution, and ensure the whole object is inside your area. What should be avoided wherever possible is creating pages where a user clearly clicks on a part of an object, but that part of the object is outside of the defined area, and so nothing happens.

# Chapter 17. Web-Application Development

## Table of Contents

Objectives.....	1
17.1 Introduction.....	1
17.2 Examples of Web applications.....	2
17.2.1 Blogs.....	2
17.2.2 Wikis.....	2
17.2.3 Sakai .....	3
17.2.4 Digital Libraries.....	3
17.3 What do Servlets do? .....	3
17.3.1 Request Methods .....	4
17.4 The structure of a Web application .....	4
17.5 Your first Web application.....	5
17.6 Handling user input.....	6
17.7 Cookies and session tracking .....	9
17.8 Problems with cookies .....	15
17.9 Response status .....	17
17.10 The Servlet packages and classes.....	19
17.10.1 javax.servlet.....	19
17.10.2 javax.servlet.http.....	20
17.11 Servlet Life-cycle.....	20

## Objectives

At the end of this unit you will be able to:

- Understand some web applications;
- What servlets do;
- Write a web application using servlet packages and classes.

## 17.1 Introduction

Web applications are computer applications that the user does not run directly on their own computer, but rather accesses through a Web browser. The GUI is provided by the Web browser (usually as some form of HTML), and all of the application's processing is done either on the browser with JavaScript, on the Web server itself, or on both. This means that Web application development is, in essence, the development of network-based applications, including server-side (i.e., on the Web server) and client- side (i.e., on the Web browser) programming. This module has so far specifically dealt with client-side development, including HTML and JavaScript, but this chapter deals with server-side development.

There are many ways to programme a Web server, including such commonly used technology as PHP and ASP.net. The traditional method of doing so is to use the Common Gateway Interface (CGI), which is a language agnostic interface for adding programmability to Web servers. It allows CGI programmes to be written in any language, and these programmes are executed by the Web server whenever a resource which they supply is requested by a Web browser.

This chapter will introduce you to the equivalent Java solution: Java Servlets. This will allow you to make use of the knowledge that you've gained in the programming module, rather than

having to learn a new language.

Servlets are Java objects which are run on a Servlet aware Web server. Each Servlet object is mapped to particular URLs on the server. Instead of simply returning a Web document when a request for an URL is made, the server instead uses the Servlet mapped to the URL to handle the request. The Servlet is free to deal with the request however it feels it should – it can communicate with databases, read or write to the file system, and create arbitrary response data to be returned to the Web browser. Typically, however, this response data is HTML, which the browser then displays for the user.

While Servlets makes use of your Java knowledge, they also have another advantage: because they are designed to replace a portion of a Web server's functionality, they will also give you insight into how a Web server deals with a request for a Web page, more so than many other Web application frameworks. This is always useful knowledge to have, no matter what Web framework you will use in the future.

To begin, you will need access to a Web server able to run Servlets. You can use any such server that you have access to, but these notes will make use of Apache Tomcat, and all of the examples will reflect this. Instructions on how to install and use Tomcat were given in Chapter 2.

## 17.2 Examples of Web applications

The Web has grown to make use of a large number of Web applications. Typical examples that you may have come across include Web-based e-mail clients, online shopping sites, blogs, wiki-wiki, and so on. Visit the sites below to see what kind of functionality they provide:

- Amazon [<http://www.amazon.com>]
- Kalahari.net [<http://www.kalahari.net>]
- E-Bay [<http://www.ebay.com>]
- GMail [<http://www.gmail.com>]
- Wikipedia [<http://www.wikipedia.org>]
- Open Street Map [<http://www.openstreetmap.org>]

### 17.2.1 Blogs

A Web log (or blog) is often described as an online diary, but they have developed into websites that may contain commentary on events and media (such as reviews), original work (such as fiction, or comics), or contain various other media besides text, with video now becoming fairly common. As such, the main feature of blogging software is that it allows the blog's author to regularly and easily add content to it, and this content is typically displayed in chronological order (much like a diary), usually from most recent to least recent. Blogs also provide some functionality to the reader, such as the ability to search through the various entries, or to leave comments for the author and other readers. Blogs can also be written by more than one person, and the software will usually keep track of which author has written which entry.

#### TO DO

Visit a blog service provider such as WordPress, LiveJournal or BlogSpot, and read about the service. Sign up for an account if you are interested to see how such software works.

### 17.2.2 Wikis

A wiki is a content management system that allows people to view, add or modify the content that it contains. Unlike a blog, which always has a specific person or group of people acting as an author, a wiki usually lets anyone edit the content, and can be described as a collaborative content creation effort by all of its users. Here are some of the common features:

Prior registration to use the service is not typically necessary. This is meant to encourage everyone to

contribute, although it does mean that vandalism can more easily occur.

The editing process is often simple, and requires no review procedure prior to publication. In other words, changes are immediately made available to everyone. Again, this is to encourage high levels of contribution.

A wiki system may keep track of the changes made to the content, allowing the users to see the individual changes and when they were made. It also makes it easy to revert content back to an older version if necessary.

One of the main advantages of a wiki is that the knowledge and content that it contains is community-based. This might not mean that the content is more accurate, just more representative of its users' views. A good wiki page should contain a wide variety of views, and as such censorship is frowned upon in many wiki sites. Another advantage is that the system allows for information to be kept up to date with minimal maintenance by any one group of people.

### **TO DO**

Visit a wiki site such as Wikipedia, a wiki-based encyclopaedia. Search for a topic and examine the page. Look closely at the discussion, edit and history tabs (on top of each page). Visit an entry on the subject that you might be an expert in. Can you see if there is anything missing? Find out how you could contribute. You could also visit the 'Help' ulink url read about different aspects of the site, such as the standards that people contributing to the site must adhere to.

## **17.2.3 Sakai**

Sakai is an online educational course management system. It allows for managing course resources (notes, exercises, answers etc.), supplies forums, chat rooms, surveys, electronic hand-in facilities, schedulers and calendars, among various other features. The system allows both teachers and students to make use of it, offering them each features appropriate to the different roles that they play. At the University of Cape, our Sakai instance is called Vula, and all students have access to it.

### **TO DO**

Visit the Vula site if you have not been regularly visiting. Do you think that such a facility has been useful to you? How could it be improved?

## **17.2.4 Digital Libraries**

A digital library, like its real-world counterpart, is a collection of books and reference materials, albeit with the exception that the content is electronic and usually provided over a network. An example of a digital library is Project Gutenberg [<http://www.gutenberg.org>], a collection of freely available books no longer under copyright protection.

Digital libraries have a number of advantages of traditional libraries, including increased capacity, lower costs and easier accessibility. A digital library can contain large volumes of information that compared to the amount of physical space that it requires. It can also be cheaper to operate one as it requires less maintenance and staff costs, and being available over a network means that it can be used to by people all over the world if the library is made available over the Internet.

However, there are also some disadvantages. For example, there are concerns that copyright issues are hampering the effectiveness of digital libraries, while the sheer volume of information has caused the traditional techniques of finding information to become inefficient. Additionally, many people find it difficult to read long documents on a computer screen, and so they are often printed out, costing money and wasting paper.

### **TO DO**

Visit a digital library such as Project Gutenberg. Find a book that you're interested in reading and see if you can download it. Read the library's conditions of use. What do you think of Project Gutenberg? Would you make use of it regularly?

## **17.3 What do Servlets do?**

Servlets replace how a Web server responds to a request for some resource. In this way they are invisible to someone visiting a Web application using a browser: the browser makes standard requests for standard Web resources, the addresses of which usually look like ordinary Web pages. The server, on the other hand, will realise that a specific address is for a Servlet in a given Web application. It will then pass control to the Servlet to handle the whole request. Any data sent to the server with the request is past to the Servlet, and the Servlet may return any data it wishes (including HTML) to the Web browser.

Because Servlets replace a portion of a Web server's functionality, we need to know a bit about HTTP, the HyperText Transfer Protocol used by Web servers and Web browsers to communicate with each other.

### 17.3.1 Request Methods

When a Web browser requests a Web page, that request is made using HTTP. Now, HTTP defines a number of actions that can be taken on a Web page. These actions are called request methods. The two that are of interest to you in this chapter are GET and POST requests. Both requests are used when data (for instance, an HTML file) is being requested from a Web server. A GET request, in particular, is used only for requesting data, although it does have some ability to pass data to a Web server as well. POST requests also request data, but importantly, data is meant to be sent to the Web server as well, and this data is meant to be processed. A POST request may result in a Web application changing its state (as it processes the data). A GET request should never change the state of a Web application. Indeed, GET is defined by HTTP to be a Safe Method, which are request methods that never modify the state of the Web application.

When an HTML file is requested by a Web browser, this request is by default made as a GET request. The important exception to this is with HTML forms, which can request that the data be sent using a POST request (by setting the method attribute of the form tag to "POST"). The Web page set as the form's action will then be requested using a POST request.

When you are writing your own Servlets, you will have the chance to directly make use of, and implement, GET and POST requests. You will notice little difference between them apart from how they transfer data from the Web browser to the Web server: a GET request will always display the sent data in the URL, while a POST request will generally never do this. Apart from this, you must always remember to use the two request types only where they are intended. Specifically, it is incorrect to update the Web server's state using a GET request.

## 17.4 The structure of a Web application

Since the Servlet API 2.2 specification, every Servlet aware Web server is required to accept Web applications in a standard format. A Servlet Web application is defined as a collection of directories and files, all of which can possibly be compressed into a single file called a Web application archive (WAR). These WAR files are a useful way to distribute your Web application once you have written it, but we will not be covering them in these notes.

The Servlet specification defines in what directories various files making up the application should be placed. If our Web application is stored in a directory called APPLICATION, then the various files should be laid out as follows:

- HTML files are stored directly in the APPLICATION directory.
- The APPLICATION/Web-INF directory contains metadata about the Web application itself, and the various Servlets which make up the application. This metadata is stored in the APPLICATION/ Web-INF/web.xml file.
- APPLICATION/Web-INF/classes contains the class files implementing the various Servlets.
- APPLICATION/Web-INF/lib contains any JAR files the application uses.

All of the following examples use the layout described above.

## 17.5 Your first Web application

Now for a Web application that dynamically creates the simple Web page previously created using Java. As before, we begin by creating a subdirectory within webapps called “hello”.

Now we create the Java class which will handle the request for the Web page. In your favourite text editor, create a new file called HelloServlet.java and add the following text:

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)

        throws ServletException, IOException
    {
        PrintWriter out = response.getWriter();

        out.println("<html>\n\t<head>\n\t\t<title>Hello!</title>\n\t</head>
");
        out.println("\t<body>\n\t\tHello, world.\n\t</body>");
    }
}
```

The javax.servlet packages are distributed with Tomcat in the servlet-api.jar file. You will find this in the “lib” subdirectory of your Tomcat distribution. Ensure that this file is in your classpath when compiling this code. If you are compiling from the command line, you can do this with the following line:

```
javac -classpath "<path to Tomcat>\lib\servlet-api.jar"
HelloServlet.java
```

Replace <path to Tomcat> with the Tomcat directory.

The first three lines import packages used in the code, and the contents of the two Servlet packages will be covered later. All of your Servlets will extend the HttpServlet class, which you will find in the javax.servlet package.

The class has one method, doGet(). This method is used to implement a GET request. This is appropriate, as when the Web browser requests the page produced by the Servlet it will be done using a GET request. doPost() is a similar method which takes the same arguments, but is called when a POST request is made. A Servlet may implement either or both methods, as it requires.

doGet() takes two arguments: an HttpServletRequest and HttpServletResponse. These encapsulate information concerning the request and the response the Servlet will make, in respect. This example is extremely simple, and only returns a message to the Web browser. It does this by getting the PrintWriter instance provided by the response object. This is the exact same type of object as System.out, only it is used to return information to a Web browser.

Next, the class writes HTML output to the PrintWriter instance. Notice that the output should produce exactly the same HTML as the previous example.

Compile the file. Inside the “hello” directory, create a directory called “Web-INF”, and inside this create a directory called “classes”. Copy the HelloServlet.class to this directory.

The Web application now needs to describe itself to Tomcat (or any other Servlet aware Web server it might run on) and also tell it what to do with the Servlet classes. This is done using the web.xml file.

## Web-Application Development

Create web.xml inside the Web-INF directory, and fill it with the following text:

```
<?xml version="1.0"?>
<!DOCTYPE Web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
  2.3//EN" "http://java.sun.com/dtd/Web-app_2_3.dtd">

<Web-app>
  <display-name>The Hello World Application</display-name>
  <description>Prints hello world!</description>

  <servlet>
    <servlet-name>hello</servlet-name>
    <description>This is a simple Servlet</description>

    <servlet-class>HelloServlet</servlet-class>

  </servlet>

  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</Web-app>
```

The first two lines declare that this file is an XML file, and gives the files document type. The document type should always be the same. The Web-app element is used to describe the application as a whole.

`<display-name>` and `<description>` describe and name the application to the Web server. Each Servlet provided by the Web application needs to be described in `<servlet>` elements. `<servlet-name>` must provide a name for the Servlet. This name does not have to be the same as the class name or the .class filename: it is the name that you will use throughout the rest of the web.xml when talking about the Servlet. `<description>` describes the Servlet in plain English. `<servlet-class>` names the actual Servlet class, in this case it is HelloServlet.

Each new Servlet must be described in its own `<servlet>` element, and each must have a unique name.

So far all that has been done is describe what Servlets are available to the Web application – now we need to tell it what to do with each Servlet. This is done with the `<servlet-mapping>` element. Each `<servlet-mapping>` element will map a particular Servlet (identified by its Servlet name) to a particular URL relative to the base Web application URL, as given by `<url-pattern>`. For instance, the above example of `<url-pattern>/</url-pattern>` says that any request made to the root address of the Web application should be handled by this Servlet. In this case, the URL would be `http://localhost:8080/hello`. On the other hand, if we had used `<url-pattern>/greetings</url-pattern>`, then the address of the Servlet would be `http://localhost:8080/hello/greetings`.

Once you have saved the web.xml file to Web-INF, and copied the .class file to the classes directory, restart Tomcat and visit `http://localhost:8080/hello`. You should see exactly the same message as the previous simple Web page produced. You can view the produced HTML by asking for the page source from your browser.

What has happened is that the web.xml has made Tomcat aware that all requests made to the root address of the Web application should be handled by the HelloWorld Servlet class. When your Web browser requests the address, it does so using the GET request method. Tomcat forwards this GET request to an instance of HelloServlet, which then prints out the HTML code returned to the Web browser.

## 17.6 Handling user input

In this example we will create a Web application which asks the user for their name, and then greets them. The greeting occurs on a separate page. Note that it is possible to do this all with JavaScript

## Web-Application Development

and the DOM in one page, on the browser. It is also possible to do this using only one page in a Web application, which we will do in the following chapter. For simplicity, however, we will use two Web pages.

As previously noted, GET and POST requests can both send data to the Web server, although both do so differently. Specifically, a GET request will encode the information in the actual URL, while a POST request hides the information in the HTTP header itself. Users will always be able to see any information sent to the server using a GET request in the URL.

Because the two methods send data differently, the data must also be obtained from each request in a different way. For our convenience, the Servlet API hides all of this detail from us: all of the data is made available through the `HttpServletRequest` object, no matter which request method has been used, and no matter how the data was sent to the server.

We begin by creating a new Web application: create a directory in the Tomcat webapps subdirectory called "greetings". Now we create a simple page that asks for the user's name:

```
<html>
  <head>
    <title>Some Greetings</title>
  </head>
  <body>
    What is your name?
    <form action="response" method="GET">
      <input type="text" name="username">
      <input type="submit">
    </form>
  </body>
</html>
```

We save this as `index.html`. Of interest is the form. Notice that it sets its method to GET, which specifies how the information in the form is communicated to the server. It also is a hint that none of the Web application's internal state will change, otherwise POST would have been used. The form's action has also been set to "response", which is the URL which the form's information will be sent to when the submit button is pressed. The form itself contains a text entry for the user to enter their name. The text entry has been called "username". The Servlet will use this name when requesting the value in the text box.

You can view this page by opening the `http://localhost:8080/greetings` in your Web browser. "greetings" is just the directory name which you created in the webapps directory. Clicking the submit button should result in a page not found error.

We will now create the Servlet providing the response page:

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;

public class ResponseServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException
    {
        String name = request.getParameter("username");
        PrintWriter out = response.getWriter();

        out.println("<html><title>Hello! " + name +
            "</title></head>"); out.println("<body>\n<h1>Hello, " +
            name + ".</h1>"); out.println("It's good to meet you.");
    }
}
```

And here is a description of it, and the Web application, in the `web.xml` file.

## Web-Application Development

```
<?xml version="1.0"?>

<!DOCTYPE Web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
  2.3//EN" "http://java.sun.com/dtd/Web-app_2_3.dtd">

<Web-app>

  <display-name>A Greetings Application</display-name>
  <description>Asks the user's name and then greets
  them.</description>

  <servlet>
    <servlet-name>ResponseServlet</servlet-name>
    <description>Performs the actual
    greeting.</description>

    <servlet-class>ResponseServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ResponseServlet</servlet-name>
    <url-pattern>/response</url-pattern>
  </servlet-mapping>
</Web-app>
```

This Servlet uses the `getParameter()` method to request the value associated with the “user\_name” entry in the form. `getParameter()` always returns a string, unless there is no entry in the form with the name passed as an argument to `getParameter()` – in this case it returns null.

Restart Tomcat. When you now open the Web application in your browser, you can enter your name and click the submit button. The application will now take you to a page greeting you by name.

This Servlet does currently have one obvious bug: it does not notice when no name has been entered, but the submit button has been pressed. Try this to see what happens. This problem can be fixed in two ways: first, JavaScript could be used in the browser to test if a name has actually been given. Secondly, the Servlet can itself test to see if a name has been given or not. We will implement the second method by modifying the Servlet's code to the following:

```
if (name.equals("")) {
    out.println("<html><title>Please enter your
    name</title></head>"); out.println("<body>\n<h1>Oh
    no!</h1>");
    out.println("You have not entered your name. Please
    press the back button o
}
else {
    out.println("<html><title>Hello! " + name +
    "</title></head>"); out.println("<body>\n<h1>Hello, " +
    name + ".</h1>"); out.println("It's good to meet
    you.");
}
```

Recompile the class and copy it into the classes subdirectory. Restart Tomcat and retest the Web application's behaviour when not entering a name.

This has just been a simple example of a Web application. Later, we will build a Servlet which will store state not only on the server, but also on the browser.

## 17.7 Cookies and session tracking

This example shows you how to track users visiting your site.

HTTP has a specific problem: when a Web server receives a request for a particular page, let us say page A, followed by another request for a page, let us say page B, the server has no easy way to tell if the same user has made both requests, or if the two requests have come from two different users. This complicates the creation of a Web application: the Web application must, in some way, be able to track its users as they move from page to page, and the application must implement a method to do this itself. There are a number of ways to do this, the most simple of which is to use HTTP cookies.

A cookie is text which the Web application sends to the Web browser. The browser stores this text and sends it to the server on every HTTP request which it makes. A cookie can generally store up to four kilobytes of any text a Web application chooses to store in it. This text can be, for example, a unique ID to identify the user. Because the cookie is returned to the server with every HTTP request, the unique ID can be used to identify which user is accessing the page, and so can be used to track the user as they move across the different pages in a Web application. This is important for, say, for letting any changes the user made to the Web application persist while the user is using the site.

The following example shows how to implement a simple hit counter. A hit counter tracks the number of unique visits that a Web page receives. Our Web counter will not count multiple visits from the same person on the same day, so that no one person can arbitrarily increase the number of hits a site seems to be receiving.

The counter we will create will use a file called "counts.txt" which will store the number of hits the page has received. To begin, create a new directory in the webapps subdirectory called "counter". We will only have one page in the Web application, which will be implemented with the following Servlet:

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;

class CounterException extends ServletException { public
    CounterException(String message)
    {
        super(message);
    }
}

public class Counter extends HttpServlet implements
    SingleThreadModel { private final String file_name =
    "counts.txt";
    private final String cookie_name = "visited"; private
    final String cookie_value = "returnSoon";

    private String getCookieValue(HttpServletRequest
        request, String name) throws CounterException
    {
        Cookie cookies[] = request.getCookies(); Cookie cookie;

        if (cookies == null)
            throw new CounterException("No cookies have been
            set");

        for (int i = 0; i < cookies.length; i++) { cookie =
            cookies[i];
            if (cookie.getName().equals(name)) return
            cookie.getValue();
        }
    }
}
```

## Web-Application Development

```
throw new CounterException("Unable to find a cookie named
    " + name);
}

private final String deleteCookieJavaScript(String
name)
{
return "var date = new Date();\n" + "document.cookie = '"
    + name + "=deleted;" + "expires=' +
    date.toGMTString() + ';';\n" +

    "alert('cookie deleted.');"
}

public void doGet(HttpServletRequest request,
    HttpServletResponse response)
{
BufferedReader file = null; int count = 0;

try {
    file = new BufferedReader(new FileReader(file_name));
    count = Integer.parseInt(file.readLine());
}
catch(IOException e) {
}
finally {
    try {
        if (file != null) file.close();
    }
    catch (IOException b) {}
}

String value = null; try {
    value = getCookieValue(request, cookie_name);
}
catch(CounterException e) {
    Cookie cookie = new Cookie(cookie_name,
    cookie_value); cookie.setMaxAge(60 * 60 * 24);
    response.addCookie(cookie);
    count += 1;
}

PrintWriter writer = null; try {
    writer = new PrintWriter(new FileWriter(file_name));
    writer.println(count);
    writer.close(); writer = null;

    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("\t<head><title>Hello, visitor number " +
count + ".</title></head>");
    out.println("<body>");
    out.println("<p>Hello, visitor number " + count +
".</p>"); out.println("<form><input type='button'
value='delete cookie' onclick=\"");
```

## Web-Application Development

```
        out.println("</body>");
        out.println("</html>");
    }
    catch (IOException e) {
    }
    finally {

        if (writer != null) writer.close();
    }

    }
}
```

The web.xml file is as follows:

```
<?xml version="1.0"?>

<!DOCTYPE Web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
    2.3//EN" "http://java.sun.com/dtd/Web-app_2_3.dtd">

<Web-app>
  <display-name>The counter application</display-name>
  <description>
    This application counts the number of unique visits it
    receives.
  </description>

  <servlet>
    <servlet-name>Counter</servlet-name>
    <description>
      A simple Servlet that uses cookies to track the
      number of unique visits it receives per day.
    </description>

    <servlet-class>Counter</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Counter</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</Web-app>
```

As usual, make sure that the .class file and web.xml are placed in their appropriate directories. Let

us look at the doGet() method. It begins like so:

```
BufferedReader file = null; int count = 0;

try {
    file = new BufferedReader(new FileReader(file_name));
    count = Integer.parseInt(file.readLine());
}
catch(IOException e) {
}
finally {
```

```

        try {
            if (file != null) file.close();
        }

        catch (IOException b) {}
    }

```

This code attempts to open the “counts.txt” file and reads in the single integer value that it contains. The integer “count” is initialized to the value zero. The code in the try block performs the actual file read. It needs to handle two exceptional situations: a.) the first time that the application is run, the file will not yet exist; b.) the Servlet may have a problem reading the file in general. Both problems will throw exceptions of type IOException, which we catch. We use a finally block to ensure that the file is always closed if it has been created. If an exception has occurred, we do nothing except let the doGet() method continue. This is perfectly fine: nearly all of the exceptions will have occurred because the “counts.txt” file does not exist. In this case, the count variable should be set to zero, which we have ensured it is when we declared it. More fine grained error control could be obtained by reporting an error for IOException (how to do this is explained later), and also catching the FileNotFoundException, in which case we do nothing and let the method continue, as above.

Now that we have read the hit count (if it so far exists), we need to decide if the counter should be incremented. This Web application uses cookies to determine if a user has previously visited the page. The following code examines if a cookie had previously been stored on the user's computer:

```

String value = null; try {
    value = getCookieValue(request, cookie_name);
}
catch(CounterException e) {
    Cookie cookie = new Cookie(cookie_name,
        cookie_value); cookie.setMaxAge(60 * 60 * 24);
    response.addCookie(cookie);
    count += 1;
}

```

getCookieValue() will throw an exception if the “visited” cookie has not been set. If this happens, the Web application creates a new cookie (which is, conveniently, an object of type Cookie) and tells the response object to store it on the users computer. The counter is also incremented. Notice that the counter is not incremented if the cookie exists. The Cookie constructor takes two arguments: the first is the cookie's name, and the second is an arbitrary text value which must be smaller than four kilobytes (a fairly substantial amount of text). By default, a cookie will only last for as long as the browser is open – the cookie is deleted when the user shuts their browser down. However, we would like the cookie to be stored on the user's computer for a whole day, so that our hit counter does not count multiple visits made on the same day. This is done using the Cookie object's setMaxAge() method. It accepts the number of seconds for which the cookie must be stored on the computer. We supply it with the number of seconds in a day (60 seconds in a minute, 60 minutes in an hour, 24 hours in a day).

The work of reading the cookie is done by the getCookieValue() method :

```

private String getCookieValue(HttpServletRequest
    request, String name) throws CounterException
{
    Cookie cookies[] = request.getCookies(); Cookie cookie;

    if (cookies == null)
        throw new CounterException("No cookies have been
        set");

    for (int i = 0; i < cookies.length; i++) { cookie =

```

## Web-Application Development

```
cookies[i];

    if (cookie.getName().equals(name)) return
    cookie.getValue();
}

    throw new CounterException("Unable to find a cookie named
    " + name);
}
```

A Web application can store multiple cookies (each with a different name) on a computer. You can gain access to the cookies using the request object's `getCookies()` method, which will return an array of cookies. `getCookieValue()` searches the array for the Cookie with the given name, and returns its value. The first time the application is run, `getCookies()` will return null, since no cookies have yet been set. In this case, or in the case where the requested cookie isn't found, we throw an exception of type `CounterException`.

Finally, we return some HTML and update the `counts.txt` file.

```
PrintWriter writer = null; try {
    writer = new PrintWriter(new FileWriter(file_name));
    writer.println(count);

    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("\t<head><title>Hello, visitor number " +
    count + ".</title></head>");
    out.println("<body>");
    out.println("<p>Hello, visitor number " + count +
    ".</p>"); out.println("<form><input type='button'
    value='delete cookie' onclick=\"");
    out.println("</body>");
    out.println("</html>");
}
catch (IOException e) {
}
finally {
    if (writer != null) writer.close();
}
```

The first few lines opens “`counts.txt`” for reading, and writes the new counts value. Notice that this application can be streamlined by only writing to the file if the count variable had been incremented.

Next, we output the actual HTML. The page prints a greeting to the visitor, and tells them what the hit count is. Also, to test the application, it provides two buttons. One of which is a submit button which you can use to reload the page. The other shows how to delete cookies by setting their maximum age to zero. This can either be done in the Web application using `setMaxAge(0)`, or it can be done as we do it here, using JavaScript. This code also catches `IOExceptions`, which can be thrown when writing to either the `counts.txt` file or the response object. The writer object is also closed in a `finally` block, to ensure that it is always closed.

The Web application should, when you load it (<http://localhost:8080/count/>), tell you the number of times the site has been visited. Pressing the reload button does not increment this number, except after you have pressed the 'delete cookie' button. You can completely reset the counter by removing the “`counts.txt`” file, which should be in Tomcat's bin directory.

Our above Servlet implements the `SingleThreadModel` interface. It does this because, typically, a Web application is used by multiple users at the same time, each of them trying to view this same page.

Tomcat spawns a thread to handle each user, and each thread will run the `doGet()` method. This makes

it possible that each thread may be trying to read and write from “counts.txt” at the same time. Clearly this could cause a problem (consider what would happen if one file reads counts.txt, updates the count variable, but before it can write the new value to the file another thread updates the file instead; now this update is going to be overwritten with an old value). The simplest solution, and the one used here, is to declare the class as implementing `SingleThreadModel`. This is an interface with no methods – all it does is to tell Tomcat to never use more than one thread at a time for this Servlet. This ensures that there is never a problem with multiple threads attempting to read and write to the file. This does have some performance implications for your Web application: it means that each user viewing the page must wait in turn for Tomcat to finish serving the page to the previous users. This is not an ideal solution when the number of users visiting the site increases. Because of this, and because Java already has its own techniques for handling synchronisation, `SingleThreadModel` has been deprecated, and no replacement has been offered. You should be using the concurrency mechanisms offered by the Java language to ensure that this type of problem does not occur – unfortunately a discussion of these methods is beyond the scope of these notes, but we do leave you with a simple example to show how this could be done:

```

        public void doGet(HttpServletRequest request,
            HttpServletResponse response)
        {
            BufferedReader file = null; int count = 0;

            synchronized (this) { try {
                file = new BufferedReader(new FileReader(file_name));
                count = Integer.parseInt(file.readLine());
            }
                catch(IOException e) {
                }
                finally { try {
                    if (file != null) file.close();
                }
            }
            catch (IOException b) {}
        }

        String value = null; try {
            value = getCookieValue(request, cookie_name);
        }
        catch(CounterException e) {
            Cookie cookie = new Cookie(cookie_name, cookie_value);
            cookie.setMaxAge(60 * 60 * 24);
            response.addCookie(cookie);
            count += 1;
        }

        PrintWriter writer = null; try {
            writer = new PrintWriter(new FileWriter(file_name));
            writer.println(count);
            writer.close(); writer = null;
        }
        catch (IOException e) {
        }
        finally {
            if (writer != null) writer.close();
        }
    }

    try {
        PrintWriter out = response.getWriter();
    }

```

## Web-Application Development

```
        out.println("<html>");
        out.println("\t<head><title>Hello, visitor number " +
count + ".</title></head>");
        out.println("<body>");
        out.println("<p>Hello, visitor number " + count +
".</p>"); out.println("<form><input type='button'
value='delete cookie' onclick=\"");
        out.println("</body>");
        out.println("</html>");
    }
    catch (IOException e) {
    }
}
}
```

Notice that all of the instructions concerning the counts.txt file has been placed into a synchronized block.

## 17.8 Problems with cookies

Cookies are not the only way which can be used to track a user as they move between the different pages of a Web application. Cookies are convenient because they can be set up so that a user can completely leave the Web application, later return to it, and continue what they were doing without any loss of information. However, the user may have disabled cookies on their browser, causing any Web application using only cookies to track its users to no longer function correctly. Two other methods that can be used are “URL rewriting” (which keeps track of a user by modifying the query string at the end of an URL) and using hidden form fields to store user data.

However, to ease Web application development, the Servlets API provides a unified interface to using Cookies and URL rewriting. This is the Session Tracking API, made available to HTTP Servlets through the HttpSession interface. HttpSession supplies a consistent way of storing and retrieving data about the user, and should generally be used in place of cookies.

Session tracking is easy to use. The first step is to ask for the current session from the HttpServletRequest object, like so:

```
HttpSession session = request.getSession();
```

This will return the current user's session object. If the user is using the Web application for the first time and no session object currently exists, a new one will be created. If you would like to ensure that a new session object is not created by default, you can use the second form of the method:

```
HttpSession session = request.getSession(false);
```

If the session does not already exist, getSession(false) returns null. getSession(true) is exactly the same as getSession().

Once you have the session object, you can store arbitrary information about the user, like this:

```
session.setAttribute("name", value);
```

“name” can be an arbitrary string that gives a name for the value. The value variable itself can be any Java object, but in these notes we will only be using strings. The value can be read like so:

```
String value = (String) session.getAttribute("name");
```

Notice that you have to cast the object returned from getAttribute().

This is all there is to using session objects. Below is an updated version of the hit counter Servlet that uses the session tracking API rather than cookies. There are a number of ways that this could be done, including one in which no attribute needs to be set. We will, however, be setting an attribute.

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {

    private final String file_name = "counts.txt"; private
    final String attribute_name = "visited"; private final
    String attribute_value = "returnSoon";

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
    {
        BufferedReader file = null; int count = 0;

        synchronized (this) { try {
            file = new BufferedReader(new FileReader(file_name));
            count = Integer.parseInt(file.readLine());
        }
        catch(IOException e) {
        }
        finally { try {
            if (file != null) file.close();
        }
        catch (IOException b) {}
        }

        HttpSession session = request.getSession(); String
        value = null;

        value = (String)
        session.getAttribute(attribute_name);
        if (value == null) {
            session.setAttribute(attribute_name, attribute_value);
            session.setMaxInactiveInterval(60 * 60 * 24);
            count += 1;
        }

        Writer writer = null; try {
            writer = new PrintWriter(new FileWriter(file_name));
            writer.println(count);
            writer.close(); writer = null;
        }
        catch (IOException e) {
        }
        finally {
            if (writer != null) writer.close();
        }
    }

    try {
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("\t<head><title>Hello, visitor number " +
        count + ".</title></head>");
        out.println("<body>");
        out.println("<p>Hello, visitor number " + count +
        ".</p>"); out.println("</body>");
        out.println("</html>");
    }
}
```

```

    }
    catch (IOException e) {
    }
    }
}

        PrintWriter writer = null; try    {
writer = new PrintWriter(new FileWriter(file_name));
writer.println(count);
writer.close(); writer = null;
    }
    catch (IOException e) {
    }
    finally {
if (writer != null) writer.close();
    }
}

try {
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("\t<head><title>Hello, visitor number " +
count + ".</title></head>");
    out.println("<body>");
    out.println("<p>Hello, visitor number " + count +
".</p>"); out.println("</body>");

    out.println("</html>");
}
catch (IOException e) {
}
}
}

```

Notice how, in the trivial case, the session tracking API is easier to use than Cookies. There are some important differences between this version and the cookie version, however: you cannot easily invalidate session tracking by simply deleting a cookie, since that cookie may not even exist; `setMaxInactiveInterval()` sets the time, in seconds, from which the user last accessed the page to when the session should be invalidated(). In contrast, `setMaxAge()` is the time in seconds from the cookie's creation. To get identical behaviour you would have to use the session object's `getCreationTime()` (which returns time in seconds) and `getLastAccessedTime()` (which returns the time in seconds since the user last accessed the Web application) methods. If the difference between those two times is larger than a full 24 hours, then the session should be manually invalidated.

## 17.9 Response status

The Cookie-based counter example had various exception handlers which should have reported an error message and stopped the execution of the `doGet()` method. However, it is not clear how an error message might be reported. For instance, when writing output to the response object, it itself may throw an `IOException` which needs to be reported in some way. The way to do this over HTTP is to use the response status. This is a number which is returned the the Web browser from the server to indicate the status of the HTTP request.

Some commonly used response statuses are:

- 200 – OK
- 403 – Forbidden
- 404 – Not found
- 500 – Internal Server Error

## Web-Application Development

When a page is successfully returned to the browser, HTTP also supplies the browser with the value 200 to indicate the the request was successful. If a request is made to a page which the user is not allowed to view, the response status 403 is given to the browser; 404 is returned when the requested page was not found. 500 is returned when there was an error processing a page. In all of these cases actual content to be displayed can also be returned. For instance, a special message indicating that the requested page was not found could be returned with the appropriate response.

Many of the response status codes are automatically sent by the Servlet based on context. For instance, if the doGet() or doPost() methods return successfully, and they have not explicitly requested a different status code, 200 will automatically be sent. For sending status codes in general, the response.setStatus(int) method can be used. The HttpServletResponse object also contains many public static final member variables that contain the appropriate values. For instance, the four codes presented above are contained in the SC\_OK, SC\_NOT\_FOUND, SC\_FORBIDDEN, SC\_INTERNAL\_SERVER\_ERROR variables.

If you want to both return HTML and set a status other than 200, be sure to set the status before you sending any output.

Errors can now be checked for and reported in our Web applications. For example, in the above counter code we could have done the following:

```
try {
writer = new PrintWriter(new FileWriter(file_name));
```

```

writer.println(count); writer.close();
writer = null;
    }
    catch (IOException e) {
response.setStatus(HttpServletResponse.SC_INTERNAL_SERVE
R_ERROR); return;
    }
    finally {
if (writer != null) writer.close();
    }

```

Which would have stopped execution of `doGet()` and notified an error. This is still not as useful as it could be: the user is not given any indication of what failed, or why. The `sendError()` method allows for this:

```

    try {
writer = new PrintWriter(new FileWriter(file_name));
writer.println(count);
writer.close(); writer = null;
    }
    catch (IOException e) { try {
response.sendError(HttpServletResponse.SC_INTERNAL_SERVE
R_ERROR,
                    )
    }

    catch (IOException e2)
    {
response.setStatus(HttpServletResponse.SC_INTERNAL_SERVE
R_ERROR); return;
    }
return;
    }
    finally {
if (writer != null) writer.close();
    }

```

Notice that `sendError` can itself throw an `IOException`; if it does, you should fall back to only setting the status.

## 17.10 The Servlet packages and classes

There are two main packages which this chapter makes use of: `javax.servlet` and `javax.servlet.http`.

### 17.10.1 javax.servlet

The `javax.servlet` package provides all of the base classes and interfaces defining both what a Servlet is, and how it interacts with the Web server that is running it. Of special note are:

#### Interfaces

- `Servlet` – This defines all the methods that a Servlet must implement. It includes methods to initialize the destroy the Servlet, and a general method (`service()`) which handles all requests made to it.

Note that classes which implement the Servlet interface need not only handle HTTP requests: they can handle any other request made to the server over other protocols. Because of this, the Servlet interface does not have the doGet() or doPost() methods, but only the general service() method. This interface also defines the init() and destroy() methods, which can be overridden to perform initialise resources used by the Servlet, and the release them when the Servlet is destroyed.

- ServletRequest – An interface which defines the methods for all objects encapsulating information about requests made to the server.
- ServletResponse – An interface defining the methods for all objects which return a response from the server.
- ServletConfig – Defines an interface used to gain access to configuration parameters which are passed to the Servlet during initialisation.

#### Classes:

- GenericServlet – This is a generic class implementing Servlet. If you wish to write Servlet's for protocols other than HTTP, the easiest way of doing so is to extend GenericServlet rather than by directly implementing the Servlet interface.
- ServletException – An exception which can be thrown when the Servlet encounters a problem of some kind.

## 17.10.2 javax.servlet.http

The javax.servlet.http package provides classes specific to handling HTTP requests. It provides the HttpServlet class used in this chapter, which implements the appropriate interfaces from javax.servlet.

#### Interfaces:

- HttpServletRequest – An extension to the ServletRequest interface for features specific to HTTP.
- HttpServletResponse – An extension to the ServletResponse interface for features specific to HTTP.
- HttpSession – Provides access to the session tracking API.

#### Classes:

- HttpServlet – An abstract class providing functionality to implement HTTP requests. Note that the service() method defined in the Servlet interface will now call doGet() and doPost(), which can each be implemented to provide behaviour to the Servlet.
- Cookie – The cookie class, which provides an interface for storing small portions of data on the user's computer.
- HttpServletRequestWrapper and HttpServletResponseWrapper – Provides an implementation of the HttpServletRequest and HttpServletResponse interfaces.

## 17.11 Servlet Life-cycle

Now that we've covered some examples and seen the interfaces and classes which make up the Servlet API, we can discuss the life-cycle of a Servlet. The Servlet life-cycle consists of the steps through which Web server places a Servlet in order to satisfy a request for a resource implemented by a Servlet. This discussion is fairly general, and applies to all Servlets, not just those extending the HttpServlet class.

## Web-Application Development

When a request for a resource implemented by a Servlet is made, the Web server does the following:

- If no instance of the particular Servlet class exists, the Web server will:
- Find the Servlet's .class file and load it.
- Instantiate the class.
- Call the classes `init()` method. `init()` is defined in the Servlet interface, so all Servlets must implement it. The method is guaranteed to be called before the Servlet handles any requests, and to only be called once. It is useful for initialising any resources the Servlet requires when handling requests, such as database connections, and so on.
- Call the Servlet's `service()` method. As previously mentioned, the `service()` method is called to handle all requests made to the Servlet. The `HttpServlet` class implements `service()` to call `doGet()` or `doPost()`, as appropriate.

When a Web server no longer requires a Servlet, it calls the Servlet's `destroy()` method. This is another method defined by the Servlet interface, and can be used to release any resources which the Servlet has acquired during its execution (for example, any database connections that it created when `init()` was called).

This Servlet life-cycle gives Servlet-based Web programmes a number of advantages over CGI programmes, which were briefly mentioned earlier in this chapter. These advantages are:

- CGI programmes are loaded in to memory whenever a new request is made for a resource implemented by the programme. When the request is completed, the programme is unloaded. Servlets always remain in memory, and so are typically faster to execute.
- Servlets share the same process as the Web server, while CGI programmes always run in their own processes. This point, along with point one above, means that Servlets remove the overhead of continuously creating new processes.
- Only a single instance of a given Servlet exists, and it answers all the requests made for a particular resource. This means that the Web server requires less memory to run, as the equivalent CGI programme will be loaded into memory multiple times in order to handle simultaneous requests.
- Just like JavaScript in a Web browser, Servlets can be run in a sandbox, giving the Servlet only restricted access to system resources, and so protecting the machine on which the Web server is running from malicious Servlets.

We can now rewrite the above counter example to make use of the Servlet life-cycle The Counter class now looks as follows:

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {

    private final String file_name = "counts.txt"; private
    final String attribute_name = "visited"; private final
    String attribute_value = "returnSoon";

    private int count = 0;

    public void init(ServletConfig config)
    {
        BufferedReader file = null; try {
            file = new BufferedReader(new FileReader(file_name));
            count = Integer.parseInt(file.readLine());
        }
    }
}
```

```
catch(IOException e) {  
}
```

```

        finally {
            try {
                if (file != null) file.close();
            }
            catch (IOException b) {}
        }
    }

    public void destroy()
    {
        PrintWriter writer = null; try {
            writer = new PrintWriter(new FileWriter(file_name));
            writer.println(count);
            writer.close(); writer = null;
        }
        catch (IOException e) {
        }
        finally {
            if (writer != null) writer.close();
        }
    }

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
    {
        HttpSession session = request.getSession(); String value
        = null;

        value = (String) session.getAttribute(attribute_name); if

        (value == null) {
            session.setAttribute(attribute_name,
                attribute_value);
            session.setMaxInactiveInterval(60 * 60 * 24);

            synchronized (this) { count += 1;
            }
        }

        try {
            PrintWriter out = response.getWriter();
            out.println("<html>");
            out.println("\t<head><title>Hello, visitor number " +
                count + ".</title></head>");
            out.println("<body>");
            out.println("<p>Hello, visitor number " + count +
                ".</p>"); out.println("</body>");
            out.println("</html>");
        }
        catch (IOException e) {
        }
    }
}

```

There are some important changes to notice: The previous versions of the Counter class read the counter.txt file when every request was made. This version of the class uses the `init()` method to read the file, and the `destroy()` method to overwrite the file. Recall that the `init()` and `destroy()` methods are called exactly once, which means that there is no need to synchronise any of the file access operations, as we have previously done. While all the code placed here in the `init()` method could be placed in the constructor, initialisation code placed in `init()` has certain advantages: first, it has access to the `ServletConfig` object, which can be used to provide configuration parameters to the Servlet. For instance, it could conceivably have contained the name of a file to store the counter information in. These configuration parameters can be set in the `web.xml` file. Second, `ServletException` objects cannot be thrown from within a constructor, but can be thrown from within `init()`.

The `doGet()` method is called multiple times, whenever a service request is made. As with previous examples, `doGet()` may be called multiple times concurrently. The only area in the code accessing and overwriting shared data occurs where the count variable is incremented. This line is wrapped in a synchronisation statement to prevent two separate calls to `doGet` from interfering with each other during. It is possible that a second call to `doGet` could increment the count variable just before the variable is printed, but this will not cause the Servlet to misbehave.

# Chapter 18. AJAX: Asynchronous JavaScript and XML

## Table of Contents

Objectives.....	1
18.1 Introduction.....	1
18.2 Initialisation.....	2
18.3 Synchronous example.....	2
18.4 A more detailed example.....	4
18.5 Getting data from a server.....	5
18.6 Sending data to a server.....	8
18.7 Performing asynchronous communication.....	10
18.8 Advantages and disadvantages of AJAX based Web pages.....	13

## Objectives

At the end of this unit you will be able to:

- Make HTTP requests using AJAX;
- Use XMLHttpRequest objects;
- Get data to and from the server using AJAX;
- Perform asynchronous communication using AJAX.

## 18.1 Introduction

With the growth of Web applications, websites have been trying to make user interaction similar to that found in desktop applications. The largest push is towards making Web interfaces more dynamic: if data changes, say a new email arrives in your inbox in GMAIL, the Web page should update and display the new email without reloading the whole page. The page should always be available to the user and always responsive to their input. While the DOM allows for such dynamic updates to occur, AJAX is an important group of technologies that allows a Web page to request more information from the server (such as an updated list of email in an inbox) without having to reload the whole page. Together, AJAX and the DOM can be used to create dynamic Web pages, and this chapter will introduce you to how that can be done.

AJAX stands for **Asynchronous JavaScript and XML**. It allows a Web page to make a request to a Web server for information using standard HTTP, but without reloading the page, and without automatically displaying the information returned from the server. These requests are all made programmatically, using JavaScript, and data communication is often done using XML, as JavaScript can easily parse this data. After receiving the data from the server, the JavaScript script can use the returned data however it wishes. The requests can also be made in a such a way that the JavaScript code does not have to wait for a reply from the server. Instead, the JavaScript code is notified when the page has finished receiving the information. In this way, the script can continue to perform useful actions while the data downloads from the server – this makes the communication asynchronous to any action that the Web page is performing.

It is important to realise that AJAX itself refers to a group of related technologies, not all of which are standardised, and not all of which need to be used at once. For instance, it can often be more convenient to communicate with the server using plain text rather than XML, and these communications need not occur asynchronously. Various other technologies are often employed in addition to those making up the AJAX

## AJAX

name itself. The original article that defined the term AJAX lists the following technologies:

- XHTML and CSS, which defines what is being displayed and how it is displayed.
- The Document Object Model, which allows us to programmatically alter the content and how it is displayed.
- XML and XSLT, which is used to transfer data between the server and Web browser (using XML), and to manipulate that data (using XSLT).
- XMLHttpRequest, which is the object used to communicate with the Web server over HTTP. This object provides the asynchronous communication abilities.
- JavaScript, which is the programming language implemented in most Web browsers, and is used to bring together all the other technologies listed above.

Most of these technologies are discussed elsewhere in these notes. The rest of this chapter will examine how XMLHttpRequest can be used to communicate with the Web server.

## 18.2 Initialisation

The first step to making an HTTP request using AJAX is to create an instance of one of the XMLHttpRequest function objects. There are two different objects that can be used, depending on the browser: Mozilla based browsers – such as Firefox – and Safari (used on OS X) both use XMLHttpRequest() objects. Internet Explorer, on the other hand, uses ActiveXObject(). An object can easily be created using the following code:

```
var httpRequest;  
if (window.XMLHttpRequest) { // Mozilla, Safari, ...  
    httpRequest = new XMLHttpRequest();  
} else if (window.ActiveXObject) { // IE  
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");  
}
```

The methods and properties associated with both of these objects are identical: the major difference between them is how they are initialised. During these notes we will speak of XMLHttpRequest to mean both the Mozilla / Safari XMLHttpRequest objects, and Microsofts XMLHttpRequest ActiveX object, unless stated otherwise.

## 18.3 Synchronous example

This simple example will show you to use XMLHttpRequest objects to request a file using JavaScript. In the same directory, create two files: one called sync-file.html, and the other called text.txt. The text.txt can contain any small amount of text that you like; we suggest using, Hello, world! This is the file that you will be requesting using the XMLHttpRequest object. Make sure that the files are in the same directory, as most browsers have security restrictions that will stop XMLHttpRequest objects from being able to access files outside of the directory that the HTML file making the request is in.

Add the following HTML to sync-file.html:

```
<html>  
  <head>  
    <title>This is a title</title>  
  
  </head>  
  <body>  
    <script type="text/javascript">  
  
      function create_request() { var httpRequest;  
        if (window.XMLHttpRequest) { httpRequest = new
```

```

                                AJAX
XMLHttpRequest();
}
else if (window.ActiveXObject) {
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
}
else
{
    document.write("Unable to create XMLHttpRequest
object.");
}

    return httpRequest;
}

var httpRequest = create_request();

httpRequest.open("GET", "file://<directory>/text.txt", false);
httpRequest.send(null);
if (httpRequest.status == 0) // For non http requests.
{
    document.write(httpRequest.responseText);
}
else
{
    document.write("There has been a problem opening the
file.");
}

</script>

</body>
</html>

```

There are a few things that you should note about this example. First, the code to create the XMLHttpRequest object has been hidden in a function called create\_request(). Note that this is not a constructor function. We will continue to use this function throughout this chapter. Second, since this code is synchronous, the request to read the document (described in detail below) returns before the page continues to be displayed by the browser. This lets us make use of document.write(), which we will not be able to use so easily after the page has finished being loaded.

Now look closely at the following lines:

```

httpRequest.open("GET", "file://<directory>/text.txt", false);
httpRequest.send(null);

```

<directory> should be replaced with the directory that the files have been saved in.

The **open** method does several things. Firstly, it does not actually open the file or Web resource requested by the method. Rather, it sets up a request for a file. The **send** method sends this request, along with any other information the programmer specifies. Only when **send** is called will any data transfer take place.

The first argument to **open** specifies the HTTP request method, as described in the previous chapter on Web application development.

The second argument is the resource that should be accessed. In this case it is a file on the hard drive, but it will usually be a resource on the Web. Note that browser security will generally stop an HTML page or JavaScript file retrieved over the Internet from accessing files on your hard drive – only html files that you saved onto your drive can generally do this. The third argument specifies whether the data should be transferred asynchronously (with the value

true), or synchronously (false). The difference to the programmer is that data transferred synchronously will be available immediately for use as soon as the **send** method is called. In fact, the **send** method will not return until all the data has been sent and received, which can cause the **send** method to take a fair amount of time to return. In turn, this can cause the JavaScript script to completely freeze while waiting for data from the server. On the other hand, if asynchronous communication is requested, the **send** method will return immediately, and all the communication will occur in the background. Unlike with synchronous communication, the data will not be ready when the **send** method returns.

## 18.4 A more detailed example

The previous example is quite trivial, since the AJAX call takes place while the page is being loaded. This example will go a step further: the AJAX request will occur after the page has completely loaded, and the page will be dynamically updated. This will be done by showing a button to the user. Pressing the button will make an AJAX request to read a file, the contents of which it will be printed on the page. While this could be done without AJAX, AJAX will allow us to do this without reloading the page.

First, create two files, one called text.txt and another called div-update.html. Text.txt should contain a small message, just as it previously did. Div-update.html should contain the following:

```
<html>
  <head>
    <title>This is a title</title>

  </head>
  <body>
    <script type="text/javascript">

      function create_request() { var httpRequest;
        if (window.XMLHttpRequest) { httpRequest = new
          XMLHttpRequest();
        }
        else if (window.ActiveXObject) {
          httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
        }
        else
        {
          document.write("Unable to create XMLHttpRequest
            object.");
        }

        return httpRequest;
      }

      function get_data_synchronously_from(file_name)
      {
        var httpRequest = create_request();

        httpRequest.open("GET", file_name, false);
        httpRequest.send(null);
        if (httpRequest.status != 0) // For non http requests.
        {
          alert("There has been an error reading the file."); return
            null;
        }
        else
        {
          return httpRequest.responseText
        }
      }
    }
  }
</script>
</body>
</html>
```

```

                                AJAX
function button_press()
{
document.getElementById("display-div").innerHTML=
get_data_synchronously_from(
}

</script>

<p>
    Click on the button below to load data from the file in the
    text box.
</p>

<form>
    <input type="text" name="file-name"
    value="file:///home/rudy/Desktop/web/
    <input type="button" value="Print the file"
    onclick="button_press();" />
</form>

<p> <b>The file currently contains: </b></p>
<div id="display-div">
    <p>The file has not yet been loaded.</p>
</div>

                                </body>
</html>

```

The example is similar to the previous one in how it handles the AJAX requests. Notice that we've encapsulated the request in the `get_data_synchronously_from()` function, which takes a file name and now returns the contents of the file. The HTML page contains a DIV with the ID "display-div". It is the DIV which we will use to display the contents of the file. The page also contains a form with two elements, a text box, which we have named "file-name", to contain the name of a file which the user wants to view, and a button. The button calls the `button_press()` function when it is clicked. This function does all of the interesting work of this example: it uses `getElementByName()` to access the text box (notice that `getElementByName()` returns an array of elements with the given name – in this case there is only one element, which is why we take the first), whose value is used as the argument to `get_data_synchronously_from()`. `getElementById()` is then used to access the DIV, and we set its contents using its `innerHTML()` attribute to the contents in the file.

The user can enter the name of any text file in the text box. The file should be in the same directory as the HTML file, otherwise some browsers will report a security error. Also, the file should have its full path specified, otherwise, again, many browsers will report a security error. When the user presses the button, the HTML page is dynamically modified (modified without reloading the page) to contain the contents of the file. Notice that the file can contain HTML code as well, and this will be displayed.

## 18.5 Getting data from a server

Our previous examples have all only read data from a file on your computer. However, AJAX is typically used to read files off of a Web server. The only substantial difference is that the a Web server will return a response status other than 0 (typically, if you recall from the previous chapter, the value 200), which is what we have been testing for in the examples.

This example will make use of the Tomcat Web server from the previous chapter. Create a directory in the Tomcat webapps subdirectory titled "random". The first thing that we will work on is a simple Servlet that will return random numbers, using Java Random objects. In the classes directory, add the class for this Java file:

```

import java.io.*; import javax.servlet.*;
import javax.servlet.http.*; import java.util.Random;

public class RandomServlet extends HttpServlet {

```

## AJAX

```
public void doGet(HttpServletRequest request, HttpServletResponse
    response)
    {
    try {
        PrintWriter out = response.getWriter(); out.println((new
            Random()).nextInt());
    }
    catch (IOException e) {
        response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    }
    }
}
```

The RandomServlet class merely instantiates the Random class, and returns the next random integer. It also handles any errors.

The appropriate web.xml file (added, as you can recall, to the WEB-INF directory), is:

```
<?xml version="1.0"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN_
    "http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>
    <display-name>Random Numbers</display-name>
    <description>
        Returns random numbers
    </description>

    <servlet>
        <servlet-name>RandomServlet</servlet-name>
        <description>
            A simple servlet that returns HTML fragments of random numbers.
        </description>

        <servlet-class>RandomServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>RandomServlet</servlet-name>
        <url-pattern>/random</url-pattern>
    </servlet-mapping>
</web-app>
```

You can test this Web application by starting Tomcat and visiting the URL: <http://localhost:8080/random/random>

The AJAX portion of this example is the index.html file, which should be added directly into the random directory:

```
<html>
    <head>
        <title>Press a button to get a number!</title>

    </head>

    <body>
        <script type="text/javascript">
```

## AJAX

```
function create_request() { var httpRequest;
    if (window.XMLHttpRequest) { httpRequest = new
        XMLHttpRequest();
    }
    else if (window.ActiveXObject) {
        httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
    }
    else
    {
        document.write("Unable to create XMLHttpRequest
            object.");
    }

    return httpRequest;
}

function get_data_synchronously_from(url_name)
{
    var httpRequest = create_request();

    httpRequest.open("GET", url_name, false);
    httpRequest.send(null);
    if (httpRequest.status != 200) // For non http requests.
    {
        alert("There has been an error reading the URL."); return
            null;
    }
    else
    {
        return httpRequest.responseText
    }
}

function button_press()
{
    document.getElementById("display-div").innerHTML=
    get_data_synchronously_from(
    }

</script>

<h1> Random numbers from the server: </h1>
<form>
    <input type="button" value="Print a number!"
        onclick="button_press();" />
</form>

<div id="display-div">
    <p>The button hasn't yet been pressed.</p>
</div>

</body>
</html>
```

This file is very similar to the previous example. Notice that `get_data_synchronously_from()` has been updated to retrieve data from an URL. Importantly, the status of the response object now returns the value 200 when the the HTTP request has been successful.

Again, there is a DIV tag with an ID whose data is replaced with the random number. The random number obtained from the RandomServlet we previously wrote.

Play with this application. You will see that the random number is always updated without reloading the page. Also notice that no matter what random number is displayed on the page, if you ask the browser to show you the page's source, you will never find the number in the source. This is because the Web page is always being dynamically updated.

## 18.6 Sending data to a server

For this example we will update the greetings Web application from the Web application chapter to use AJAX.

Create a new directory in webapps called "ajax-greetings". We will use the following class, which you should compile and place in the classes directory:

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*; import java.util.Random;

public class ResponseServlet extends HttpServlet
{
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
    {
        try {

            String name = request.getParameter("name"); PrintWriter out =
                response.getWriter();

            if (name == null) {
                out.println("No name has been given."); return;
            }

            out.println(getGreeting() + name);
        }
        catch (IOException e) {
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERR
                OR);
        }
    }

    String getGreeting()
    {
        switch((new Random()).nextInt(5)) { case 0:
            return "Hello, "; case 1:
            return "Nice to meet you, "; case 2:
            return "Hi, "; case 3:
            return "Yo, "; case 4:
            return "Hey, ";
        }
        return "Hello, ";
    }
}
```

This class accepts POST requests only. It is expecting to be passed the user's name in the "name" parameter, and then uses the getGreeting() method to find a random way of greeting the person. The Servlet then returns a single line, greeting the user.

The appropriate web.xml file is:

## AJAX

```
<?xml version="1.0"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
  2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>A Greetings Application</display-name>
  <description>Asks the user's name and then greets
  them.</description>

  <servlet>
    <servlet-name>ResponseServlet</servlet-name>
    <description>Performs the actual greeting.</description>

    <servlet-class>ResponseServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ResponseServlet</servlet-name>
    <url-pattern>/response</url-pattern>
  </servlet-mapping>
</web-app>
```

The index.html file, which should be placed directly in the ajax-greetings directory, follows:

```
<html>
  <head>
    <title>Some Greetings</title>
  </head>
  <body>
    <script type="text/javascript">

      function create_request() { var httpRequest;
        if (window.XMLHttpRequest) { httpRequest = new
          XMLHttpRequest();
        }
        else if (window.ActiveXObject) {
          httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
        }
        else
        {
          document.write("Unable to create XMLHttpRequest object.");
          return httpRequest;
        }
      }

      function get_data_synchronously_from(file_url, variables)
      {
        var httpRequest = create_request();

        httpRequest.open("POST", file_url, false);
        httpRequest.setRequestHeader("Content-Type", "application/x-
        www-form-urlencoded");
        httpRequest.send(variables);
        if (httpRequest.status != 200) // For non http requests.
        {
          alert("There has been an error reading the URL. " +
            httpRequest.status); return null;
        }
      }
    </script>
  </body>
</html>
```

## AJAX

```
{
  return httpRequest.responseText
}

function button_press()
{
  document.getElementById("greeting-div").innerHTML=
  get_data_synchronously_f
}

</script>

<div id="greeting-div"> What is your name?
</div>
<form>
  <input type="text" name="user_name">
  <input type="button" value="Say Hello!"
  onclick="button_press();" >
</form>
</body>
</html>
```

The page appears as before, with a text box for the user to enter their name and a button to press to return a greeting. The page displays this greeting in a DIV with an ID, which is how we have dynamically updated Web pages previously in this chapter. The bulk of the work is done in the `get_data_synchronously_from()` function, which has been updated to accept a second argument, “variables”. This is a list of data to be sent to the server, and is formatted as a list of names and their values: `name=value&name2=value2&name3=value3` and so on. In this case, we only provide one name / value pair when calling this function. We do this in the `button_press()` function, which takes the value from the text box.

`get_data_synchronously_from()` also makes a call to `setRequestHeader()` to tell the server that data has been sent. If this call is not made, the server will ignore all data sent to it by this request. Next, `variables` is passed to the `send` function. The rest of the function operates exactly as before.

You can now test the application. You should notice that the Web application greets the user without reloading the page. The application also uses multiple, random greetings.

## 18.7 Performing asynchronous communication

When creating an interactive application it is important to not let the application unduly freeze while it is performing an action. This is equally true when making a request to a server using an `XMLHttpRequest` object: when making a request from the server, the interface should remain usable, responding to actions from the user, even if only to say that the Web application is currently busy. One important place that special care may be required is when the `XMLHttpRequest` results in enough information being transferred from the server that there is a noticeable delay in the browser receiving it. For instance, this can easily occur when downloading a large document to embed in the Web page. But remember that often there can be delays even when transferring very small amounts of information, since some users may be connecting to the Web application using slower connection methods, or data may have been lost during transit.

Application freezing usually occurs when `send` is used and the `open` method – which you should remember is called just before calling `send` – was called with the third argument set to `false`. In this case `send` does not return until it has finished receiving all the data being sent from the server. If it takes ten seconds to receive the data, it takes ten seconds for the `send` function to return from its call. And for those ten second the browser will not perform any other action, including accepting user input.

We call the form of communication where we wait for the server to finish sending the browser information before continuing to execute the Web application synchronous communication. As previously mentioned, AJAX allows for asynchronous communication: while the data is being transferred, the Web application can continue to execute, and can accept user input or perform other functions.

## AJAX

Asynchronous communication is fairly easy to perform: instead of passing false to the open function, we pass true to indicate that we want to communicate asynchronously with the server. When send is then called, send will immediately return. In the synchronous case this would have meant that communication with the server is now complete, and any information that it was sending to the browser is now available. In the asynchronous case this is not true. Rather, we supply the XMLHttpRequest object with a function which it should call when the state of the communication changes. For instance, the method is called when the communication begins, and when the communication ends. The XMLHttpRequest object has a property called readyState, which indicates when the communication has finished: if it holds the integer value 4, then communication is done, and the data is ready to be used. The data may then be read from the XMLHttpRequest object in the usual way.

The example for this section is a rewrite of the Random Web application we developed above so that it performs its request for a random number asynchronously. Begin by copying the contents of the random directory to a new directory in webapps called async-random.

Edit the index.html file so that it contains the following:

```
<html>
<head>
  <title>Press a button to get a number!</title>

</head>
<body>

<script type="text/javascript"> function create_request() {
  var httpRequest;
  if (window.XMLHttpRequest) { httpRequest = new XMLHttpRequest();
  }
  else if (window.ActiveXObject) {
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
  }
  else
  {
    document.write("Unable to create XMLHttpRequest object.");
  }

  return httpRequest;
  }

function get_data_asynchronously_from(file_url, variables,
event_handler)
{

  var httpRequest = create_request();

  httpRequest.onreadystatechange = function() {
    event_handler(httpRequest);
  }

  if (variables != null)
  {
    httpRequest.open("POST", file_url, true);
    httpRequest.setRequestHeader("Content-Type", "application/x-
www-form-urlencoded");
    httpRequest.send(variables);
  }
  else
  {
    httpRequest.open("GET", file_url, true);
    httpRequest.send(null);
  }
}
}
```

## AJAX

```
function is_data_ready(request)
{
    return request.readyState == 4;
}

function get_value(request)
{
    if (request.status != 200) // For non http requests.
    {
        alert("There has been an error reading the URL. " +
            request.status); return null;
    }
    else
    {
        return request.responseText
    }
}

function button_press()
{
    function set_value(request)
    {
        if (is_data_ready(request))
            document.getElementById("display-
                div").innerHTML=get_value(request);
    }

    get_data_asynchronously_from("http://localhost:8080/random/rando
        m", null, set
    }

</script>

<h1> Random numbers from the server: </h1>
<form>
    <input type="button" value="Print a number!"
        onclick="button_press();" />
</form>
<div id="display-div">
    <p>The button hasn't yet been pressed.</p>
</div>
</body>
</html>
```

The first large change is the **get\_data\_asynchronously\_from()** function, which now accepts a third argument, which will be the name of a function to be used in the XMLHttpRequest **onreadystatechange** event handler: this is the function that will handle the data retrieved from the server. The function passed to **get\_data\_asynchronously\_from()** should always accept one argument, which will be the XMLHttpRequest object which is retrieving the data. This allows the function to test if the server has finished sending the data, and to obtain the data from the object. The first two arguments to **get\_data\_asynchronously\_from()** are exactly the same as previously.

After creating the XMLHttpRequest object, the first thing that **get\_data\_asynchronously\_from()** does is to set the event handler which will be called when the data is ready. Note that the event handler for **onreadystatechange** does not accept any arguments, so a new function is created which calls the event\_handler function passed to **get\_data\_asynchronously\_from()** and passes it the XMLHttpRequest object.

The next if statement determines if any data (contained in the **variables** variable) is being sent to the server, and decides on whether POST or GET request methods should be used.

## AJAX

Finally, **send** is called, either passing the data or passing null, as necessary.

There are two new functions: **is\_data\_read()** takes a XMLHttpRequest object and returns true if communications with the server has completed; otherwise it returns false.

**get\_value()** takes an XMLHttpRequest object and is exactly the code from the old **get\_data\_synchronously\_from()** functions that tests to see what status the server has given for the HTTP request, and returns the data.

**button\_press()** has also been updated. First, it creates a function called **set\_value**, which is the event handler we will pass to **get\_data\_asynchronously\_from**. This function first tests to see that the XMLHttpRequest object has completed communications with the server (since the event handler is called whenever communication state with the server changes, and not only when communication completes). It then retrieves the sent data, and updates the Web page as in the previous example.

# 18.8 Advantages and disadvantages of AJAX based Web pages

AJAX is increasingly being used on the Web today. The main reasons for this are:

- AJAX allows a Web page to only reload those portions which have changed, rather than reloading the whole page. This can substantially reduce the amount of bandwidth that the Web application requires.
- Because AJAX allows for asynchronous communication, and for only reloading a portion of a Web document, it allows the Web application to be more frequently available to the user, since page reloads are reduced, and the user can continue using the application while data is being transmitted.

However, AJAX does have disadvantages, which need to be weighed up against the advantages, and mitigated where possible:

- Dynamically created pages, such as most of those using AJAX, often renders the browser's history functionality useless. If the user moves away from a page, and then presses the back button to return to it, the page is not necessarily shown in the same state as when the user left it. It may be useful to expose all distinct functionality as separate URLs, so that the user can visit them using the back and forward buttons, or the browser's history list.
- Similar to the above problem, it can be difficult to bookmark any useful page and return to it, since what the user was actually interested in bookmarking had been loaded dynamically.
- Search engines indexing a website are not likely to find any of the dynamically generated content, which will effect how the site appears in search engines' rankings.