



NATIONAL OPEN UNIVERSITY OF NIGERIA

SCHOOL OF SCIENCE AND TECHNOLOGY

COURSE CODE: CIT 462

COURSE TITLE: WEB SERVER TECHNOLOGY

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island, Lagos

Abuja Office
5 Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja

e-mail: centralinfo@nou.edu.ng

URL: www.nou.edu.ng

Published by
National Open University of Nigeria

Printed 2014

ISBN: 978-058-600-8

All Rights Reserved

CONTENTS	PAGE
Introduction.....	iv
What you will Learn in this Course.....	iv
Course Aims	iv
Course Objectives.....	iv
Working through the Course.....	v
Course Materials.....	v
Study Units.....	vi
Textbooks and References.....	vi
Assignment File.....	vii
Assessment.....	viii
Tutor-Marked Assignment.....	viii
Final Examination and Grading.....	viii
Presentation Schedule.....	viii
Course Marking Scheme.....	ix
Course Overview.....	ix
How to Get the Most from this Course.....	x
Facilitators/ Tutors and Tutorials.....	xi
Summary.....	xi

INTRODUCTION

You are expected to read this study guide carefully at the start of this semester. It contains important information about this course. If you need more clarifications, please consult one of the teaching staff. This course material will provide you an in-depth knowledge you will need in order to complete the course.

The code for this course is CIT 462 and the course title is Web Server Technology. It is a two - credit unit course for students studying towards acquiring a Bachelor of Science in Computer Science and other related disciplines.

The course is divided into 4 modules and 11 study units. It will first present a brief review of the concepts of web server technology and then deal with the different stages involved in developing good and functional skills of web server technology applications. The course went further to introduce you to different ways of using different languages to develop good web server applications. This course also introduces you to other knowledge that will enable you have proper understanding of web server technology.

The Course Guide therefore gives you an overview of what the course is all about; the textbooks and other materials to be referenced; what you expect to know in each unit; and how to work through the course material.

You are welcome to web server technology where you will be taught how to use different web technologies to design good web applications.

WHAT YOU WILL LEARN IN THIS COURSE

The overall aim of this course is to equip you with the concepts and skills of writing a good web applications with database implementation.

COURSE AIM

This course aims to introduce you to the basics, concepts and features of web server technologies. The knowledge will provide you with effective skills for writing good web applications using web server technologies.

COURSE OBJECTIVES

It is important to note that each unit has specific objectives. You should study them carefully before proceeding to subsequent units. Therefore, it may be useful to refer to these objectives in the course of your study of the unit to assess your progress. You should always look at the unit objectives after completing a unit. In this way, you can be sure that you have done what is required of you by the end of the unit. It is hoped that by the time you complete this course you should be able to:

- analyse web server technology
- write and manipulate different client scripting languages
- write and manipulate different server side programming languages
- implement a database operations and integrate a web application with database.

WORKING THROUGH THE COURSE

For you to complete this course successfully, you are required to study all the units, the recommended text books, and other relevant materials. Each unit contains some self-assessment exercises and tutor-marked assignments, and at some point in this course, you are required to submit the tutor-marked assignments. There is also a final examination at the end of this course. Stated below are the components of this course and what you have to do

COURSE MATERIALS

The major components of the course are:

1. Course Guide
2. Modules
3. Study Units
4. Text Books
5. Assignment File
6. Presentation Schedule

In order to complete the learning successfully, you should:

- apply yourself to undergoing this course
- do not regard any aspect of this course as simplistic, difficult or complicated

- discard any previous biases about other related course(s) when you read this course
- regard the present course as an opportunity to engage in life-long learning as well as enhance your skills.

STUDY UNITS

This course is divided into four modules. The first module has one unit and deals with web server technology. The second has four units and deals with client scripting languages. The third has three units that deals with server sides programming languages. The last module which has three units is database operational. The modules are thus:

Module 1

Unit1 Web- Based Application

Module 2

Unit 1 XHTML
Unit 2 CSS
Unit 3 Java Script
Unit 4 XML

Module 3

Unit 1 Perl CGI
Unit 2 PHP
Unit 3 ASP

Module 4

Unit 1 Implementation of Database Internals
Unit 2 An Overview of Database Operations
Unit 3 Integrated Web Application

TEXTBOOKS AND REFERENCES

These textbooks and especially the internet resource links will be of enormous benefit to you in learning this course: [Academic Tutorials \(2008\). Academic Tutorials.Com.](#)

- Ailamaki, A. *et al.* (2003). *Exposing Undergraduate Students to Database System Internals*. SIGMOD Record, Vol. 32, No. 3.
- Alex, Chaffee (n.d.). What is a Web Application (or "webapp")?
<http://www.jguru.com/faq/view.jsp?EID=129328>
- ASPTutorial.info, *Active Server Pages Tutorials for Beginners*, 2002.
[Online] www.asptutorial.info.
- Benoit, Marchal (2000). 'XML By Example'.
www.ebay.com/ctg/Applied-XML
- Castagnetto, J. *et al.* (1994). *Professional PHP Programming*. Wrox Press: USA.
- CSS Beginner's Guide (n.d.). [www.marcomm.upm.edu.my/dokumen/13032_CSS_Beginner\[1\].pdf](http://www.marcomm.upm.edu.my/dokumen/13032_CSS_Beginner[1].pdf)
- CSS tutorial (2007). 'Tutorialpoint.com'.
www.tutorialspoint.com/css/index.htm.
- David, Gowans (2001). JavaScript
<http://www.freewebmasterhelp.com/tutorials/xhtml>
- Erik, T. R. (n.d). *Creating Self-Describing Data*. (2nd ed.).Covers W3C XML Schema
- Hamilton, Jacqueline D. (2004). *CGI Programming 101. Perl for the World Wide Web*,
CGI 101.COM
- Hellerstein, J. *et al.* (2007). *Architecture of a Database System, Foundations and Trends in Databases*.
- HTML.net, *ASP Tutorial*, 2005. [Online] <http://html.net/tutorials/asp/>
- Itzik Ben-Gan. (2008). *Microsoft SQL Server T-SQL Fundamentals*. Microsoft Press.
- Jen, O. M. (2007). *CSS Basics and Samples*.
<http://meiert.com/en/blog/20070922/user-agent-style-sheets/>
- Mary Had a Little Lamb (2008). *Beginner CSS Tutorial*.
<http://www.gobookeee.com/css-tutorials-for-beginners/>
- Marty, Hall (2009). *JavaScript*.

<http://courses.coreservlets.com/CourseMaterials/pdf/ajax/JavaScript-Core.pdf>

ASSIGNMENT FILE

The assignment file will be given to you in due course. In this file, you will find all the details of the work you must submit to your tutor for marking. The marks you obtain for these assignments will count towards the final mark for the course. Altogether, there are 15 tutor-marked assignments for this course.

ASSESSMENT

There are two aspects to the assessment of this course. First, there are tutor-marked assignments; and second, the written examination.

Therefore, you are expected to take note of the facts, information and problem solving gathered during the course. The tutor-marked assignments must be submitted to your tutor for formal assessment, in accordance to the deadline given. The work submitted will count for 30% of your total course mark. At the end of the course, you will need to sit for a final written examination called pen-on-paper (POP). This examination will account for 70% of your total score.

TUTOR- MARKED ASSIGNMENTS (TMAs)

There are four tutor-marked assignments (TMAs) called TMA 1, TMA 2, TMA 3 and TMA 4 as a (CBT) computer-based test in this course.

They will be available on your portal as at when due and so you should be conversant with your portal. They must be submitted to your tutor for formal assessment, in accordance to the deadline given. Each TMA carries 10 marks and the best three TMA scores out of the four submitted will be selected and count for you. This means that the TMA's submitted will count for 30% of your total course mark.

FINAL EXAMINATION AND GRADING

The final examination for CIT 462 will last for a period of 2 hours and have a value of 70% of the total course grade. The examination will consist of questions which reflect the self-assessment exercise and tutor-marked assignments that you have previously encountered. Furthermore, all areas

of the course will be examined. It would be better to use the time between finishing the last unit and sitting for the examination, to revise the entire course. You might find it useful to review your TMAs and comment on them before the examination. The final examination covers information from all parts of the course.

PRESENTATION SCHEDULE

The presentation schedule included in this Course Guide provides you with important dates for completion of each tutor-marked assignment. You should therefore endeavour to meet the deadlines.

COURSE MARKING SCHEME

The following table shows the course marking scheme:

COURSE MARKING SCHEME

Assessment	Marks
Tutor-Marked Assignments (TMAs)	4 Assignments with 20 questions each, 30% for the Best 3TMAs Total = 10% X 3 = 30%
Final Examination	70% of overall course marks
Total	100% of course mark

COURSE OVERVIEW

This table indicates the units, the number of weeks required to complete them and the assignments.

COURSE ORGANISER

Unit	Title of the work	Weeks Activity	Assessment (End of Unit)
	Course Guide		
Module 1			
Unit 1	Web Based Application	Week 1	Assessment 1
Module 2			
Unit 1	XHTML	Week 2	Assessment 2

Unit 2	CSS	Week 3	Assessment 3
Unit 3	JavaScript	Week 4	Assessment 4
Unit 4	XML	Week 5	Assessment 5
Module 3			
Unit 1	Perl CGI	Week 6	Assessment 6
Unit 2	PHP	Week 7	Assessment 7
Unit 3	ASP	Week 8	Assessment 8
Module 4			
Unit 1	Implementation of Database Internal	Week 9	Assessment 9
Unit 2	An Overview of Database Operations	Week 10	Assessment 11
Unit 3	Integrated Web Application	Week 12	Assessment 12

HOW TO GET THE MOST FROM THIS COURSE

In distance learning, the study units replace the university lecturer. This is one of the great advantages of distance learning: you can read and work through specially designed study materials at your own pace, and at a time and place that suit you best. Think of it as reading the lecture instead of listening to a lecturer. In the same way that a lecturer might set you some readings to do, the study units tell you when to read your set books or other materials, and when to undertake computing practical work. Just as a lecturer might give you an in-class exercise, your study units provide exercises for you to do at appropriate points.

Each of the study units follows a common format. The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next is a set of learning objectives. These objectives let you know what you should be able to do by the time you have completed the unit. You should use these objectives to guide your study. When you have finished the unit, you must go back and check whether you have achieved the objectives.

If you make a habit of doing this you will significantly improve your chances of passing the course.

The main body of the unit guides you through the required reading from other sources. This will usually be either from your set books or from a reading section.

Self-tests are interspersed throughout the units. Working through these tests will help you to achieve the objectives of the unit and prepare you for the assignments and the examination.

Remember, if you run into any trouble, contact your tutor.

READ THIS COURSE GUIDE THOROUGHLY

Organise a study schedule. Refer to the course overview for more details. Note the time you are expected to spend on each unit and how the assignments relate to the units.

Once you have created your own study schedule, do everything you can to stick to it. The major reason students fail is that they get behind with their course work. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.

FACILITATORS/TUTORS AND TUTORIALS

There are hours of tutoring provided in support of this course. You will be notified of the dates, time and location of these tutorials, together with name and phone number of your tutor, as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, and keep a close watch on your progress and on any difficulties you might get. Your tutor will provide assistance to you during the course.

SUMMARY

Module 1 introduces you to web-based application, history, uses, benefits and drawbacks of web based application. Module 2 presents you a step-by-step approach to writing a good client scripting languages and how to use different scripting languages like XHTML, CSS, JavaScript and XML Scripting language.

Module 3 shows you a step-by- step approach to writing a good server side programming and also languages on how to use words like: Perl CGI, PHP and ASP.

Module 4 shows you an overview of database operations, implementation and its integration with web applications. It is recommended that you draw up a schedule on how to accomplish the goals of the course.

I wish you success as you read this course and hope that you will find it interesting and useful.



**MAIN
COURSE**

CONTENTS	PAGE
Module 1	1
Unit 1 Web- based Application.....	1
Module 2	7
Unit 1 XHTML.....	7
Unit 2 CSS.....	19
Unit 3 JAVASCRIPT	44
Unit 4 XML.....	79
Module 3	93
Unit 1 PERL CGI	93
Unit 2 PHP	103
Unit 3 ASP	111
Module 4	118
Unit 1 Implementation of Database Internals...	118
Unit 2 An Overview of Database Operations...	126
Unit 3 Integrated Web Application	135

MODULE 1

Unit 1 Web- Based Application

UNIT 1 WEB- BASED APPLICATION

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 What is a Web-Based Application?
 - 3.2 History of Web-Based Applications
 - 3.3 Structure of Web-Based Applications
 - 3.4 Uses of Web-Based Applications
 - 3.5 Benefits and Drawbacks of Web-Based Applications
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 Reference/Further Reading

1.0 INTRODUCTION

A web-based application is any application that is usable only with an active Internet connection and that uses HTTP as its primary communications protocol.

Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client. The ability to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. Common web applications include webmail, online retail sales, electronic banking platforms amongst others.

2.0 OBJECTIVES

At the end of the unit, you should be able to:

- define and identify web-based applications
- explain the architectural components of web-based applications

- outline useful applications of web-based applications.

3.0 MAIN CONTENT

3.1 What is a Web-Based Application?

A web-based application refers to any program that is accessed over a network connection using http, rather than existing within a device's memory. Web-based applications often run inside a web browser.

However, web-based applications may also be client-based, where a small part of the program is downloaded to a user's desktop, but processing is done over the internet on an external server.

3.2 History of Web-Based Applications

In the early days of the web each individual web page was delivered to the client as a static document, but the sequence of pages could provide an interactive experience, as user input is returned through web form elements embedded in the page mark-up.

In 1995, Netscape introduced a client-side scripting language called JavaScript allowing programmers to add some dynamic elements to the user interface that ran on the client side. So instead of sending data to the server in order to generate an entire web page, the embedded scripts of the down loaded page can perform various tasks such as input validation or showing/hiding parts of the page.

In 1996, Macromedia introduced flash, a vector animation player that could be added to browsers as a plug-in to embed animations on the web pages. It allowed the use of a scripting language to program interactions on the client side with no need to communicate with the server.

In 1999, the "web application" concept was introduced in the Java language in the server specification version 2.2. at that time both JavaScript and xml had already been developed, but Ajax had still not yet been coined and the xml http request object had only been recently introduced on internet explorer 5 as an active object.

In 2005, the term Ajax was coined, and applications like g-mail started to make their client sides more and more interactive. A web page script is able

to contact the server for storing/retrieving data without downloading an entire web page.

In 2011, HTML5 was finalised, which provides graphic and multimedia capabilities without the need of client side plunging. HTML5 also enriched the semantic content of documents.

3.3 Structure of Web-Based Applications

Applications are usually broken into logical layers called "tiers", where every tier is assigned a role. While traditional applications are typically made up of a single tier which resides on the client machine, web applications lend themselves to n-tiered approach by nature.

Though many variations are possible, the most common structure is the 3-tiered application. In its most common form, the 3 tiers are as follows (from top to bottom):

- Presentation tier
- Application logic tier
- Storage tier.

A web browser is the 1st tier (presentation), an engine using some dynamic web content technology (such as ASP, PHP, Python, Ruby-on-Rails, or Struts) is the middle tier (application logic), and a database is the 3rd tier (storage). The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface.

For more complex applications, a 3-tier solution may fall short, and it may be beneficial to use an n-tiered approach, where the greatest benefit is breaking the business logic, which resides on the application tier, into a more fine-grained model. Another benefit may be adding an integration tier that separates the data tier from the rest of tiers by providing an easy-to-use interface to access the data. For example, the client data would be accessed by calling a "list clients ()" function instead of making an SQL query directly against the client table on the database. This allows the underlying database to be replaced without making any change to the other tiers.

There are some who view a web application as 2-tier architecture. This can be a "smart" client that performs all the work and queries a "dumb" server, or a "dumb" client that relies on a "smart" server. The client would handle the presentation tier, the server would have the database (storage tier), and the business logic (application tier) would be on one of them or on both. While this increases the scalability of the applications and separates the display on the database, it still does not allow the true specialisation of the layers, so most applications will outgrow this model.

3.4 Uses of Web-Based Applications

An emerging strategy for application software companies is to provide web access to software previously distributed as local applications.

Depending on the type of application, it may require the development of an entirely different browser-based interface, or merely adapting an existing application to use different presentation technology. These programs allow the user to pay a monthly or yearly fee for use of a software application without having to install it on a local hard drive. A company which follows this strategy is known as an application service provider (ASP), and ASPs are currently receiving much attention in the software industry.

Security breaches on these kinds of applications are a major concern because it can involve both enterprise information and private customer data. Protecting these assets is an important part of any web application and there are some key operational areas that must be included in the development process.

In cloud computing, web applications are software as a service (SaaS). These are business applications provided as SaaS for enterprises for fixed or usage dependent fees. Other web applications are offered free of charge, often generating income from advertisements shown in web application interface.

Beyond business uses, web-based applications have been adopted for a variety of useful purposes including audio and video communication, file sharing, word processing, spreadsheet, project management, network and server administration and scientific research. A notable example of a web-based application is an internet operating system or internet OS. This term is used in the computer industry to refer to any type of operating system designed to run all of its applications and services through an internet client, generally a web browser. The advantages of such an OS would be that it runs on a thin client, allowing cheaper, more easily manageable computer systems; it requires all applications to be designed on cross-platform, open standards; and would not tie a user's applications, documents, and preferences to a single computer, but rather place them on the cloud. The internet OS has also been promoted as the perfect type of platform for SaaS. An example currently in use is the Google chrome OS.

3.5 Benefits and Drawbacks of Web-Based Applications

The benefits of web applications are as follows:

- Web applications do not require any complex "roll out" procedure to deploy in large organisations. A compatible web browser is all that is needed;
- Browser applications typically require little or no disk space on the client;
- They require no upgrade procedure since all new features are implemented on the server and automatically delivered to the users;
- Web applications integrate easily into other server-side web procedures, such as email and searching;
- They also provide cross-platform compatibility in most cases (i.e., windows, mac, linux, etc.) because they operate within a web browser window;
- With the advent of HTML5, programmers can create richly interactive environments natively within browsers. Included in the list of new features are native audio, video and animations, as well as improved error handling.

As highlighted above, the use of web applications provides numerous benefits. However, it also presents drawbacks some of which are listed below:

- In practice, web interfaces, compared to thick clients, typically force significant sacrifice to user experience and basic usability;
- Web applications absolutely require compatible web browsers. If a browser vendor decides not to implement a certain feature, or abandons a particular platform or operating system version, this may affect a huge number of users;
- Standards compliance is an issue with any non-typical office document creator, which causes problems when file sharing and collaboration becomes critical;
- Browser applications rely on application files accessed on remote servers through the internet. Therefore, when connection is interrupted, the application is no longer usable;
- Since many web applications are not open source, there is also a loss of flexibility, making users dependent on third-party servers, not

allowing customisations on the software and preventing users from running applications offline (in most cases);

- They depend entirely on the availability of the server delivering the application. If a company goes bankrupt and the server is shut down, the users have little recourse. Traditional installed software keeps functioning even after the demise of the company that produced it (though there will be no updates or customer service);
- The companies that provide web-based applications can theoretically track anything the users do. This can cause privacy problems.

4.0 CONCLUSION

As technology advances, applications are likely to become more web-based either partially or entirely. While this may prove to be useful to technologically-advanced regions, it poses a serious challenge in areas where internet connectivity is either non-existent or too slow to handle application demands. That being said, web applications have provided a level of collaboration, connectivity and interactivity that traditional offline applications would never have been able to provide.

5.0 SUMMARY

In this unit, we began by examining the definition of a web-based application. We examined the history of the development of web-based applications and some associated technologies. We also discussed the architecture of web-based applications as well as their uses. Finally, we concluded the unit by taking a look at some of the benefits as well as drawbacks of using web-based applications.

6.0 TUTOR-MARKED ASSIGNMENT

- i. Briefly explain how the google chrome OS works.
- ii. Write brief history of web-based applications
- iii. What are the benefits and drawbacks of web-based applications

7.0 REFERENCE/FURTHER READING

Alex Chaffee (n.d.). 'What is a web application (or "webapp")?'
<http://www.jguru.com/faq/view.jsp?EID=129328>.

MODULE 2

Unit 1	XHTML
Unit 2	CSS
Unit 3	Java Script
Unit 4	XML

UNIT 1 XHTML

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	What is XHTML?
3.2	What do you Need?
3.3	XHTML was Created for Two Main Reasons
3.4	Rules in Writing XHTML
3.4.1	The Root Element
3.4.2	The XMLN Declaration
3.4.3	Doctype
3.4.4	Document Formation
3.4.5	Case
3.4.6	Closing Tag
3.4.7	Quote
3.4.8	Minimisation
3.4.9	The Id Attribute
3.4.10	Script Tag
3.4.11	Nesting
3.5	HTML/XHTML Code Tags
4.0	Conclusion
5.0	Summary
6.0	Tutor- Marked Assignment
7.0	References/Further Reading

1.0 INTRODUCTION

XHTML (eXtensible Hyper Text Markup Language) is a family of XML markup languages that mirror or extend versions of the widely-used Hypertext Markup Language (HTML), the language in which web pages are written. While HTML (prior to HTML5) was defined as an application of standard generalised markup language (SGML), a very flexible markup

language framework, XHTML is an application of XML, a more restrictive subset of SGML. Because XHTML documents need to be well-formed, they can be parsed using standard XML parsers unlike HTML, which requires a lenient HTML specific parser.

XHTML is really the future of the internet. It is the newest generation of HTML (coming after HTML 4) It has many new features that make it work the same way as XML. In this material I will explain how XHTML differs from HTML and how you can update your pages to support it.

It is necessary to have a basic understanding of HTML before reading this tutorial as it deals with the differences between XHTML and HTML.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- define XHTML
- differentiate between the XHTML and the HTML tags
- write your XHTML pages backwards compatible with HTML.

3.0 MAIN CONTENT

3.1 What is XHTML?

XHTML stands for eXtensible Hyper Text Markup Language and is a cross between HTML and XML. XHTML is HTML with XML qualities, which are XML with HTML written into the document type definition (DTD). This XHTML tutorial demonstrates how to code web pages using XHTML. It assumes that you're already comfortable with HTML. XHTML is almost identical to HTML with only a handful of differences. XHTML is slightly stricter than HTML and requires a little more attention when coding. XML has become the chosen language for the web's future.

XHTML was developed by the W3C to help web developers make the transition from HTML to XML. So if you want to learn XML, XHTML is a good place to start.

3.2 Tools Needed

Very few tools are needed to create, test, and deploy an XHTML document.

- **Text Editor**

Almost any text editor will do (windows notepad, for example), the best software to use is also the most basic. Notepad is a perfectly suitable tool for writing XHTML. Some programs that are specifically designed to create web pages, such as dream weaver, have special XHTML writing functions that allow you to create straight code. There are also programs such as HTML kit that function as "code insertion" programs where the user doesn't necessarily write the code, but rather instructs the program to insert blocks of HTML or XHTML. Even though these programs make writing code easier, it is still necessary to know the code. In this material we suggest that you use notepad or a similar program.

- **Web Browser**

All web browsers have provided adequate support for XHTML. Some browsers, such as mozilla's firefox, offer better support than others.

- **FTP Client**

Software designed to facilitate the transfer of files from a local computer to a remote server. Many operating systems (including microsoft windows) come bundled with basic FTP clients. This tool will not be required for these lessons.

3.2 XHTML Objectives

1. To create a stricter standard for making web pages, reducing incompatibilities between web browsers
2. To create a standard that can be used on a variety of different devices without changes.

The great thing about XHTML, though, is that it is almost the same as HTML, although it is much more important that you create your code correctly. You cannot make badly formed code to be XHTML compatible. Unlike with HTML (where simple errors (like missing out a closing tag) are ignored by the browser), XHTML code must be exactly how, it is specified to be. This is due to the fact that browsers which are in hand-held devices and so on. They do not have the power to show badly formatted pages so

XHTML makes sure that the code is correct and that it can be used on any type of browser.

XHTML is a web standard which has been agreed by the W3C and, as it is backwards compatible, you can start using it in your web pages now. Also, even if you don't think it is really necessary to update to XHTML, there are three very good reasons to do so. They are as follows:.

- It will help you to create better formatted code on your site
- It will make your site more accessible (both in the future and now due to the fact that it will also mean you have correct HTML and most browsers will show your page better)
- XHTML is planned to replace HTML 4 in the future.

There is really no excuse not to start writing your web pages using XHTML as it is so easy to pick up and will bring many benefits to your site.

3.3 Rules in Writing XHTML

Here are the rules for writing in XHTML

- The root element of the document must be HTML
- The root element must contain a XMLNS declaration for the XHTML namespace
- Must have a doctype declaration, and it must come before the HTML element
- All documents must be properly formed
- All XHTML tags are lower case
- Close all tags (even empty elements)
- Attribute values must be quoted
- Minimisation is forbidden
- The id attribute replaces the name attribute
- The language attribute of the script tag is deprecated
- Proper nesting of tags.

Here's a more detailed explanation of the above XHTML rules.

3.4.1 The Root Element

Root element is the element that has no parents, so the only root element in XHTML is the <html> element that is the first element in XHTML coding.

3.4.2 The XMLNS Declaration

All XHTML documents must have an XMLNS declaration at the doctype for the XHTML namespace (i.e. <http://www.noun.edu.ng/2013/xhtml>).

Example

```
<html xmlns="http://www.noun.edu.ng/2013/xhtml" xml:lang="en" lang="en">
```

3.4.3 Doctype

All XHTML documents must have a doctype declaration. The document must include the usual html, head, title, and body elements. There must be a doctype declaration in the document prior to the root element. The public identifier included in the doctype declaration must reference one of the three DTDs found in [DTDs](#) using the respective formal public identifier. The system identifier may be changed to reflect local system conventions.

The doctype should be the very first line of your document and should be the only thing on that line. You do not need to worry about these confusing older browsers because the doctype is actually a comment tag. It is used to find out the code which the page is written in, but only by browsers/validators which support it, so this will cause no problems.

The first change which will appear on your page is the doctype. When using HTML it is considered good practice to add a doctype to the beginning of the page. This was optional in HTML, XHTML requires you to add a doctype. There are three available for use.

Strict - This is used mainly when the markup is very clean and there is no 'extra' markup to aid the presentation of the document. This is best used if you are using cascading style sheets for presentation.

Example

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.noun.edu.ng/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Transitional - This should be used if you want to use presentational features of HTML in your page.

Example

```
<!DOCTYPE html
  PUBLIC "-//noun//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.noun.edu.ng/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Frameset - This should be used if you want to have frames on your page.

Example:

```
<!DOCTYPE html
  PUBLIC "-//nou//DTD XHTML 1.0 Frameset//EN"
  "http://www.noun.edu.ng/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

3.4.4 Document Formation

After the doctype line, the actual XHTML content can be placed. As with HTML, XHTML has <html><head><title> and <body> tags but, unlike with HTML, they must all be included in a valid XHTML document. The correct setup of your file is as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.noun.edu.ng/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html>
<head>
<title>Page Title</title>
OTHER HEAD DATA
</head>
<body>
CONTENT
</body>
</html>
```

It is important that your document follows this basic pattern. This example uses the transitional doctype but you can use either of the others (although frames pages are not structured in the same way).

3.4.5 Case

All XHTML tags should be lower case. (HTML tags on the other hand, can be upper or lower case). Probably the biggest changes in XHTML are not only the tags you use but, the way in which you write them must be correct. Luckily the major change can be easily implemented into a normal HTML document without much problem.

In XHTML, tags must always be lower case. This means that

are all incorrect tags and must not be used. The font tag must now be used as follows:

Example

Wrong

XHTML Tutorial

Right

XHTML Tutorial

If you are not writing your code, but instead use a WYSIWYG editor, you can still begin to drift your documents to XHTML by setting the editor to output all code in lower case. For example, in dream weaver 4 you can do this by going to:

Edit -> Preferences -> Code Format

And making sure that Case for Tags is set to: <lowercase> and also that Case for Attributes is set to: lowercase="value"

3.4.6 Closing Tags

All tags should have an opening tag and a closing tag. If it is a tag that does not have a closing tag (for example, the img tag, or the hr tag, use a space and forward slash (/) before the > sign. (Note that the XHTML specification does not require a space before the "/" but this is required to conform to some browsers).

All tags in XHTML must be closed. Most tags in HTML are already closed (for example <p></p>, ,) but there are several which are standalone tags which do not get closed. The main three are:

<hr>

There are two ways in which you can deal with the change in specification. The first way is quite obvious if you know HTML. You can just add a closing tag to each one.

Example:

```
<img></img>  
<br></br>  
<hr></hr>
```

Although you must be careful that you do not accidentally place anything between the opening and closing tags as this would be incorrect coding. The second way is slightly different but will be familiar to anyone who has written WML (wireless mobile Language is an XML language used to specify content and user interface for WAP devices like PDA and mobile phones). You can include the closing in the actual tag.

```
<br/>  
<img/>  
<hr/>
```

Example

Wrong

```

```

Right

```

```

This is probably the best way to close your tags, as it is the recommended way by the W3C who set the XHTML standard. There may be space before the `/>` in HTML but this is not actually necessary in the XHTML specification (you could have `
`) but the reason why I have included it is that, as well as being correct XHTML, it will also make the tag compatible with past browsers. As every other XHTML change is backwards compatible, it would not be very good to have a simple missed out space causing problems with site compatibility.

In case you are wondering how the `` tag works if it has all the normal attributes included, here is an example:

```

```

Again, notice the space before the `/>`

In this part of the XHTML study material, I will show you the changes to HTML attributes in XHTML. HTML attributes are the extra parts you can add onto tags (such as `src` in the `img` tag) to change the way in which they

are shown. There are four changes to the way in which attributes are changed.

3.4.7 Quotes

In XHTML, all attribute values must be quoted. (In HTML on the other hand, you could get away without quoting attribute values).

Wrong

```

```

Right

```

```

3.4.8 Minimisation

Minimisation is forbidden. You should explicitly state the attribute and the value it contain.

Wrong

```
<option selected>
```

Right

```
<option selected="selected">
```

3.4.9 The Id Attribute

The id attribute replaces the name attribute. Instead of using name=, use id=.

Wrong

```

```

Right

```

```

3.4.10 Script Tag

The language attribute of the script tag is deprecated.

Wrong

```
<script language="javascript" type="text/javascript" >
    document.write("Feel free to link to this XHTML Tutorial!");
</script>
```

Right

```
<script type="text/javascript" >
    document.write("Feel free to link to this XHTML Tutorial!");
</script>
```

3.4.11 Nesting

All the XHTML tags must be nested properly otherwise your document will be assumed as a wrong XHTML document. Following is the example showing this difference:

Wrong

```
<b><i> This text is bold and italic</b></i>
```

Right

```
<b><i> This text is bold and italic</i></b>
```

Apart from these strict rules, you can code your XHTML pages just like you have been coding HTML pages.

3.5 HTML/XHTML Code Tags

HTML/XHTML code consists of two types of tags. There are

- i. The container tags examples are: <h1>...</h1>, <a href...>... etc.
- ii. The single part tags, often called "non-container" tags examples are:
, <hr> etc.

Some tags have optional end tags examples are: <p>, , <tr>, <td>, <th> etc.

1. Container tags have an opening part and a closing part with the "stuff" that the tag controls inside. An example is the tag that creates bold text, . is the opening part and is the closing part.

If we write a code snippet it might look like this, "This is bold" and you can see the bold text in between the tag parts.

2. An example of a single part tag is the line break, `
` which merely stops a line of text at a specific point. This is used where a line must end after a particular word.

Because web pages will have lots of code, trying to remember which tag has been opened or closed can be extremely difficult. The best technique for writing code is to make both the opening and closing parts at the same time, and then insert the material between the tags. This prevents the problems that arise when a tag is accidentally left unclosed and is the technique that will be used in this material. Single part tags can be inserted as required anytime.

4.0 CONCLUSION

This material has shown you most of the changes between HTML and XHTML which are very few. This will help you to update your site anytime you visit it next to make it XHTML compatible. It will not only make your site 'future-proof' but will also mean that you will have more correct code and should have fewer browser incompatibility problems.

5.0 SUMMARY

The tutorial suggests that HTML should be "an application of XML". The purpose is to tighten HTML's programming standards to make them compliant with XML. XML is very specific, one thing means one thing and in HTML it is not so specific. For example:

- Tags can be in caps or not.
- TEXT AREA boxes require end tags, yet text boxes do not.
- Tags can end in any order regardless of how they were placed.

You can easily convert from html to XHTML by doing the following:

1. Add an XHTML `<!DOCTYPE>` to the first line of every page
2. Add an XMLNS attribute to the html element of every page
3. Change all element names to lowercase
4. Close all empty elements
5. Change all attribute names to lowercase
6. Quote all attribute values

6.0 TUTOR-MARKED ASSIGNMENT

- i. Define XHTML

- ii. Describe the needs for effectively coding in XHTML.
- iii. Show with example how XHTML elements can be properly and improperly nested within each other.
- iv. Explain with code snippet examples the rules in writing XHTML
- v. Discuss the two main reasons for creating XHTML.
- vi. Differentiate between the HTML and XHTML
- vii. Describe how code written in html can be convert to XHTML code
- viii. Convert the following HTML Code snippet to XHTML code
 - a. `<i>This text is bold and italic</i>`
 - b. `<p>This is a paragraph`
`<p>This is another paragraph`
 - c. A break: `
`
A horizontal rule: `<hr>`
An image: ``
 - d. `<table WIDTH="100%">`
 - e. `<input checked>`
`<input readonly>`
`<input disabled>`
`<option selected>`

Correction:

1. `<i>This text is bold and italic</i>` (In XHTML, all elements must be properly nested within each other)
2. `<p>This is a paragraph</p>`
`<p>This is another paragraph</p>` (XHTML elements must always be closed)
3. A break: `
`
A horizontal rule: `<hr />`
An image: `` (empty elements must Also be closed)
4. `<table width="100%">` (attribute values must be quoted)
5. `<input checked="checked">`
`<input readonly="readonly">`
`<input disabled="disabled">`
`<option selected="selected">` (attribute minimisation is forbidden)

7.0 REFERENCES/FURTHER READING

David, Gowans (2001).

<http://www.freewebmasterhelp.com/tutorials/xhtml/>

Roger, Lipera (2008). *Introduction to HTML/XHTML V.1*. Interactive MediaCenter lipera@uamail.albany.edu <http://library.albany.edu/imc/>

Simon, Jessey (2012). *The HTML Element*. <http://jessey.net/blog/>
W3C liability (2007). XHTML <http://www.w3.org/Consortium>.

UNIT 2 (CSS)

CONTENTS

- 1.0 Introduction
 - 1.1 Which Software Do I Need?
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 What is CSS (Cascading Style Sheets)?
 - 3.2 Advantages of CSS (Cascading Style Sheets)
 - 3.3 Cascading Style Sheets Syntax
 - 3.4 Applying Style Sheets
 - 3.4.1 Embedding Cascading Style Sheets in the <head>
 - 3.4.2 Inline Style Sheets in a HTML tag
 - 3.4.3 External Style Sheets
 - 3.4.4 Importing Style Sheets
 - 3.5 Cascading Style Sheets Class and Id
 - 3.6 Cascading Style Sheets Browser Compatibility
 - 3.6.1. Cascading Style Sheet Compatibility
 - 3.6.2. Netscape and Style Sheets
 - 3.7 CSS, SPAN and DIV
 - 3.7.1 Span
 - 3.7.2 Div
 - 3.8 Cascading Style Sheets and HTML Forms
 - 3.9 Alternative Style Sheets and Link
 - 3.10 Cascading Style Sheets and Borders
 - 3.10.1 The Border Commands
 - 3.10.2 Setting the Border Commands
 - 3.10.3 The Border Style Commands
 - 3.10.4 Div
 - 3.11 Cascading Style Sheets and HTML Lists
 - 3.11.1 List- Style- Sheets
 - 3.11.2 List- Style- Position
 - 3.11.3 List- Style- Type
 - 3.11.4 List- Style
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Reading

1.0 INTRODUCTION

Cascading Style Sheets (CSS) is a [style sheet language](#) used for describing the [presentation semantics](#) (the look and formatting) of a document written in a [markup language](#). CSS defines how HTML elements are to be displayed. CSS is easy to learn and understand, it provides powerful control over the presentation of an HTML document.

Most commonly, CSS is combined with the Mark-up languages such as HTML or XHTML. CSS is an extension to basic HTML that allows you to style your web pages.

In this unit, you will learn about cascading style sheets and why you should use them.

Cascading style sheets (CSS) is a simple and powerful language for adding style to web documents. A CSS (cascading style sheet) file allows you to separate your web sites (X) HTML content from its style. As always you use your (X)HTML file to arrange the content, but all of the presentation (fonts, colours, background, borders, text formatting, link effects & so on...) are accomplished within a CSS. At this point you have some choices of how to use the CSS, either internally or externally.

This course material covers both versions CSS1 and CSS2 and teaches you CSS starting from basic concepts to advanced concepts. So start now and read through it to become master in CSS.

Before you begin, it is important that you know windows or unix. A working knowledge of windows or unix makes it much easier to learn HTML.

You should be familiar with:

- Basic word processing using any text editor.
- How to create directories and files.
- How to navigate through different directories.
- Basic understanding on Internet browsing using a browser like Internet Explorer or Firefox etc.
- Basic understanding on developing simple Web Pages using HTML or XHTML. You are expected to have covered the unit one before moving on to this unit.

1.1 Which Software Do I Need?

Please avoid using software such as front-page, dream weaver or Word with this tutorial. Advanced software will not help you learn CSS. Instead, it will limit you and slow down your learning curve significantly. All you need is a free and simple text editor such as notepad.

Microsoft Windows Notepad: is usually located in accessories in the start menu under programs. Alternatively, you can use a similar text editor e.g. pico for linux or simple text for macintosh. A simple text editor is ideal for learning html and css because it doesn't affect or change the codes you type. That way, your successes and errors can only be attributed to yourself not the software. A browser and a simple text editor is all you need.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- open a web page
- handle CSS to control the style and layout of multiple web pages
- control the colour of the text, the style of fonts, the spacing between paragraphs, and also the columns sized and laid out
- describe what background images or colours are used, as well as a variety of other effects.

3.0 MAIN CONTENT

3.1 What is CSS (Cascading Style Sheets)?

CSS is the acronym for: 'Cascading Style Sheets' is a simple design language intended to simplify the process of making web pages presentable.

An example of a style change would be to make words bold. In standard HTML you would use the tag like so:

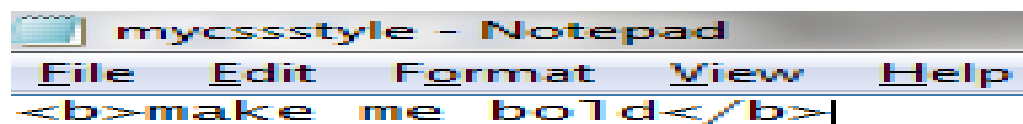


Fig. 1.1: Style Change that Makes Word Bold

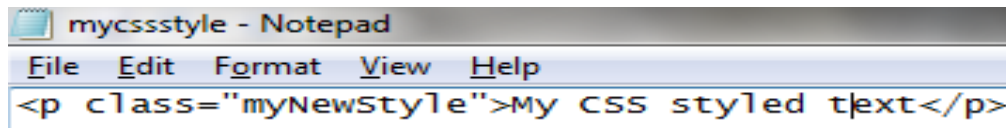
This works fine and there is nothing wrong with it per se, except that now if you wanted to say change all your text that you initially made bold to

underlined, you would have to go to every spot in the page and change the tag.

Another disadvantage can be found in this example: say you wanted to make the above text bold, make the font style verdana and change its colour to red; you would need a lot of code wrapped around the text:

```
<font colour="#FF0000" face="Verdana, Arial, Helvetica, sans-serif">
<Strong>this is text</strong></font>
```

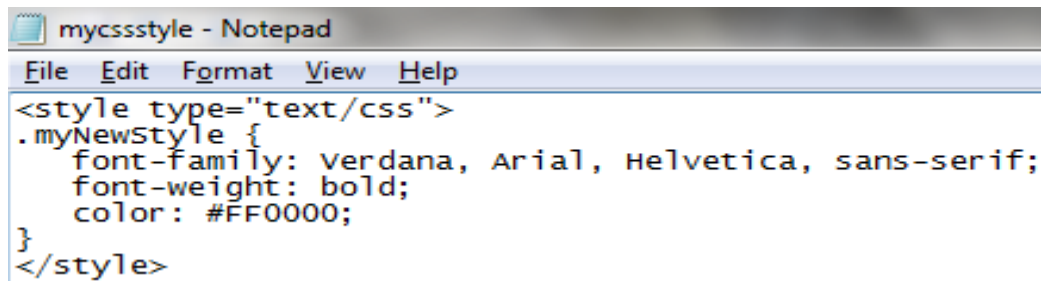
This is verbose and contributes to making your HTML messy. With CSS, you can create a custom style elsewhere and set all its properties, give it a unique name and then 'tag' your HTML to apply these stylistic properties:


 A screenshot of a Notepad window titled "mycssstyle - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The text in the editor is:


```
<p class="myNewStyle">My CSS styled text</p>
```

Fig. 1.2: CSS Style

And in between the tags at the top of your web page you would insert this CSS code that defines the style we just applied:


 A screenshot of a Notepad window titled "mycssstyle - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The text in the editor is:

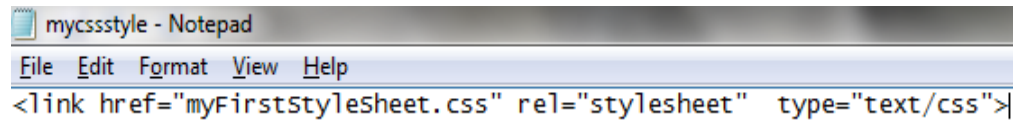

```
<style type="text/css">
.myNewStyle {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  font-weight: bold;
  color: #FF0000;
}
</style>
```

Fig. 1.3: CSS Style

In the above example we embed the CSS code directly into the page itself. This is fine for smaller projects or in situations where the styles you're defining will only be used in a single page. There are many times when you will be applying your styles to many pages and it would be a hassle to have to copy and paste your CSS code into each page.

Besides the fact that you will be cluttering up your pages with the same CSS code, you also find yourself having to edit each of these pages if you want to make a style change. Like with JavaScript, you can define/create

your CSS styles in a separate file and then link it to the page you want to apply the code to:

A screenshot of a Notepad window titled 'mycssstyle - Notepad'. The window has a menu bar with 'File', 'Edit', 'Format', 'View', and 'Help'. The text area contains the following HTML code:

```
<link href="myFirstStyleSheet.css" rel="stylesheet" type="text/css">
```

Fig. 1.4: CSS Link Style

The above line of code links your external style sheet called 'myFirstStyleSheet.css' to the HTML document. You place this code in between the <head></head> tags in your web page.

3.2 Advantages of CSS

- **CSS Saves Time** - You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many web pages as you want.
- **Pages Load Faster** - If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply to all the occurrences of that tag. So less code means faster download times.
- **Easy Maintenance** - To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
- **Superior Styles to HTML** - CSS has a much wider array of attributes than HTML so you can give far better look to your HTML page in comparison of HTML attributes.
- **Multiple Device Compatibility** - Style sheets allow content to be optimised for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
- **Global Web Standards** - Now HTML attributes are being deprecated and it is being recommended to use CSS. So it is a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

3.3 Cascading Style Sheets Syntax

Style sheets work by controlling the value (s) of the property(s) of a selector

CSS example of syntax:

Selector {property: value ;}

Almost any html tag can be used as a selector in the example below the <TT> tag is our selector, the property is colour and the value is red.

Example: TT {colour: red ;}

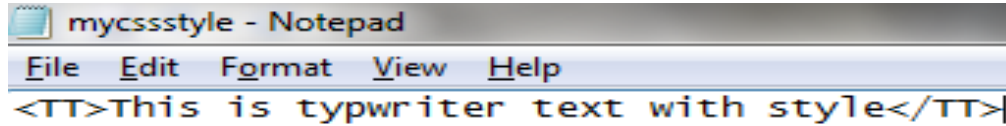


Fig. 1.5: CSS <TT> Tag

For clarity style sheets are usually written out like this...

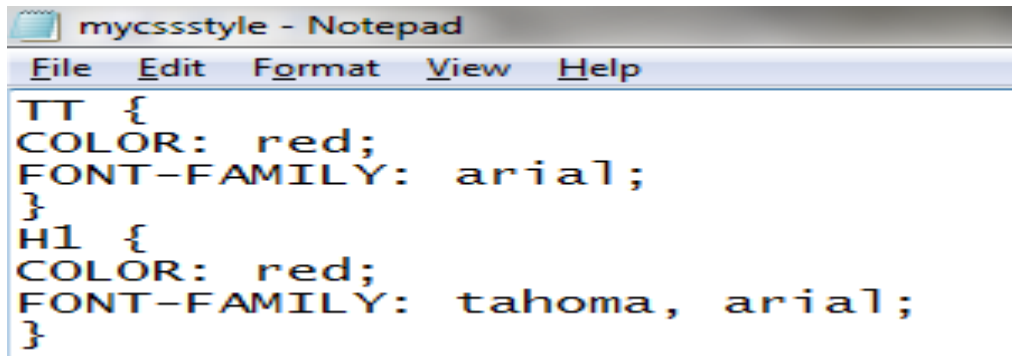


Fig. 1.6: CSS <TT> Tag Format

In the style sheet example above the value of the H1 font-family property is set to 'tahoma' with 'arial' as the back-up font. If 'tahoma' is unavailable on the user's computer then the selector (H1) will have arial as it is default font.

3.4 Applying Style Sheets

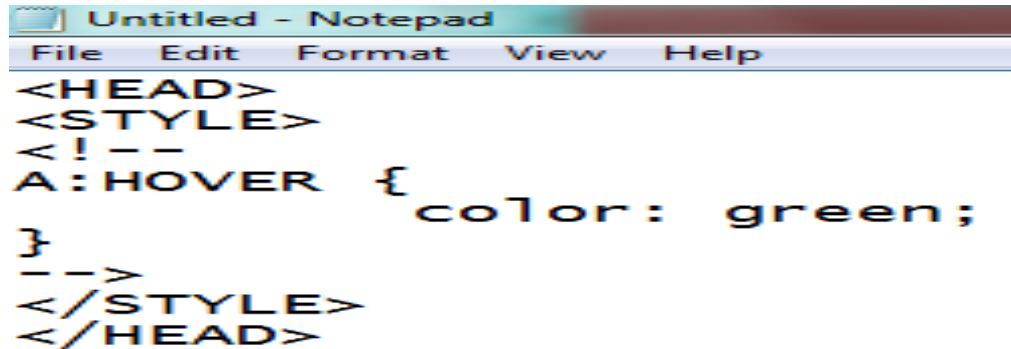
This material shows four main methods of applying style sheets to a HTML document, these are

1. Embedding style sheets in the <head></head> part of a webpage
2. Inline style sheets in a html tag
3. Linking to an external style sheet
4. Importing style sheets

3.4.1 Embedding Cascading Style Sheets in the <head> Tag

In this example we change the colour of a link we can suggest the colour of links, visited (vlink) and active links (alink) in the body tag `<body link="blue" vlink="purple" alink="red">`

Adding the style sheet below to the head of a html document will cause links to be green when the mouse cursor hovers over them.



```

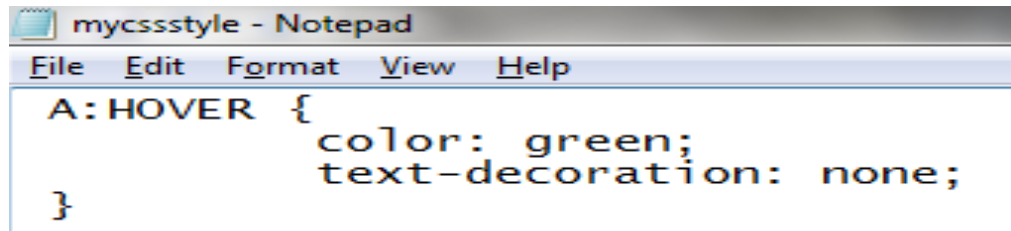
Untitled - Notepad
File Edit Format View Help
<HEAD>
<STYLE>
<!--
A:HOVER {
        color: green;
}
-->
</STYLE>
</HEAD>

```

Fig. 1.7: Embedding CSS in the <head> Tag

But style sheets can do much more than that, if you are using internet explorer you may have noticed that the links not only change colour but the underline is also removed.

Here is an example of how to achieve this



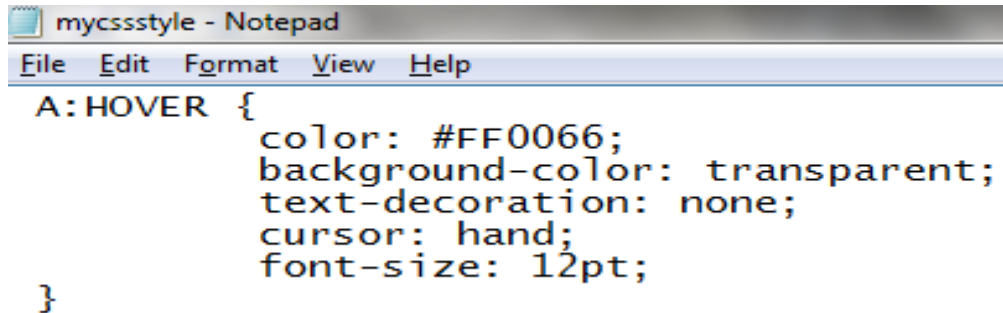
```

mycssstyle - Notepad
File Edit Format View Help
A:HOVER {
        color: green;
        text-decoration: none;
}

```

Fig. 1.8: CSS Tag

We can use the style sheet to control the font size, font colour, background colour and cursor (in Internet Explorer)



```

mycssstyle - Notepad
File Edit Format View Help
A:HOVER {
    color: #FF0066;
    background-color: transparent;
    text-decoration: none;
    cursor: hand;
    font-size: 12pt;
}

```

Fig. 1.9: CSS Controlling Font

3.4.2 Inline Style Sheets in a HTML Tag

Style properties can easily be included in individual HTML tags like this...

```
<A HREF="Yahoo" STYLE="colour: red; text-decoration: underline
;">Yahoo</A>
```

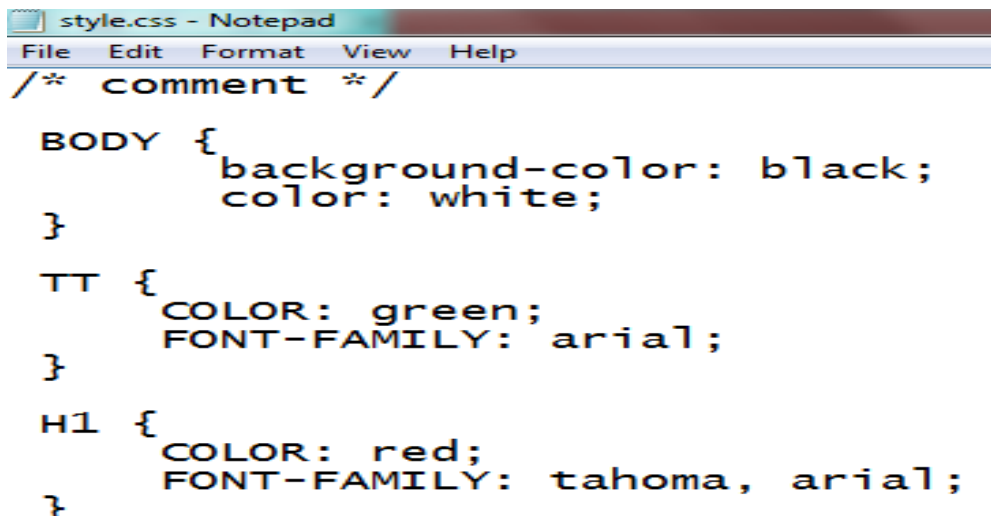
```
<TT STYLE="colour: yellow; font-family: arial; background-colour:
purple ;">
```

This is typewriter text with an inline style sheet

```
</TT>
```

3.4.3 External Style Sheets

The ability to use an external style sheet adds a tremendous amount of flexibility to html documents, because any amount of pages can link to the same .CSS document a uniform look can be applied to an entire website the benefits of an external style sheet are that file size is reduced because individual html documents do not require a style sheet to be embedded as well as the obvious saving in time, an external style sheet can be written with most simple text editors and should be saved as whatever-you-want. CSS but for the sake of argument let call it style.css



```

style.css - Notepad
File Edit Format View Help
/* comment */

BODY {
    background-color: black;
    color: white;
}

TT {
    COLOR: green;
    FONT-FAMILY: arial;
}

H1 {
    COLOR: red;
    FONT-FAMILY: tahoma, arial;
}

```

Fig. 1.10: External Style Sheets

Notice that comments can be added to the style sheet between the `/*slash|star*/` but html tags should not be included

```
/* comments go in here */
```

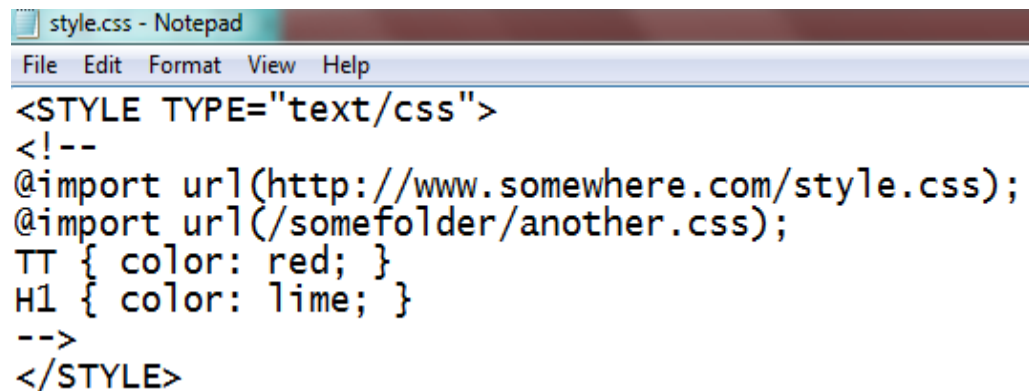
Below is an example of the link that would be used for style.css and is placed in the HEAD part of a html document

```
<HEAD>
<LINK REL="STYLESHEET" TYPE="text/css" HREF="style.css">
</HEAD>
```

3.4.4 Importing Style Sheets

An example of the import statement is included here:

A style sheet may be imported with the `@import` statement. The `@import` statement may be used in a .css file or inside the `<style>` element



```
style.css - Notepad
File Edit Format View Help
<STYLE TYPE="text/css">
<!--
@import url(http://www.somewhere.com/style.css);
@import url(/somefolder/another.css);
TT { color: red; }
H1 { color: lime; }
-->
</STYLE>
```

Fig. 1.11: Importing Style Sheets

The style sheet example above shows that CSS rules may be included but the `@import` statement must occur at the start of the style sheet.

Rules specified in the style sheet override rules in the imported style sheets, even if one of the imported style sheets contained:

```
TT {colour: blue ;}
```

Typewriter text would still be rendered in red

The order in which the style sheets are imported determines how they cascade. In the example above, if the first imported style sheet (style.css) specifies that PRE elements be shown in red and the second (another.css) style sheet specifies that PRE elements be shown in purple, in this case the second imported style sheet (another.css) overrides the first style sheet (style.css) and PRE elements would be rendered in purple.

3.5 Cascading Style Sheets Class and Id

How to give one HTML tag more than one appearance with the use of class and id

1. Cascading Style Sheets and Class

Previously we looked at applying css values to tag (selector) properties

```
Selector {property: value}
```

```
TT {colour: red ;}
```

This page describes how style sheets can alter those values

The easiest way to explain class is to think of it as making-up a descriptive word to describe the effect we want to create the example below forces any text between the <TT></TT> tags to be red in colour and 8pt in size

```
TT {colour: red; font-size: 8pt ;}
```

```
<TT>this is typewriter text with style</TT>
```

However if the <TT></TT> tags are required to be a colour other than red we can do this with a 'class' (a descriptive word preceded by a dot or period) the example below makes text between the <TT></TT> purple and 12pt in size

```
purple {colour: purple; font-size: 12pt;}
```

```
<TT>this is typewriter text with style</TT>
```

```
<TT class="purple">this is typewriter text with class</TT>  
Other tags can share a class  
<B class="purple">this is bold text with class</B>
```

2. Cascading Style Sheets and ID

Style sheet ID and class are almost the same thing except where a class is declared by being preceded by a dot (period) an ID is preceded by a # hash mark

```
#goblue {colour: blue; font-size: 12pt;}
```

```
<TT>This is typewriter text with style</TT>  
<TT class="purple">This is typewriter text with class</TT>  
<TT ID="goblue">This is typewriter text with ID</TT>  
<B ID="goblue">This is bold text with ID</B>
```

The ability to reuse the same class or id in multiple HTML tags can be a great timesaver.

3.6 Cascading Style Sheets Browser Compatibility

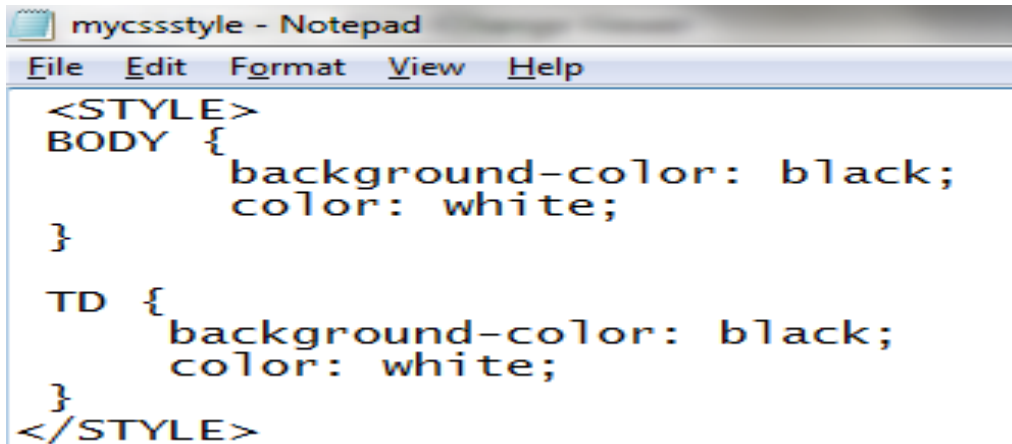
The trouble with style sheets, how to avoid problems with cascading style sheets, the rule when using CSS is to avoid depending on a style sheet completely, and use HTML to mimic the style sheet as closely as possible.

3.6.1. Cascading Style Sheet Compatibility

Although style sheets (CSS1) have been supported since netscape 4.0 and microsoft internet explorer 4 (partial CSS1 support from IE3) users do have the option of simply 'turning off' style sheets and this should be considered when using style sheets.

It should also be noted that if a netscape user disables "JavaScript" (and some of them do!) CSS seems to be disabled automatically.

- a. <TD> tag the same properties as the <BODY> tag



```
<STYLE>
BODY {
    background-color: black;
    color: white;
}

TD {
    background-color: black;
    color: white;
}
</STYLE>
```

Fig. 1.12: <TD> Tag the Same Properties as the <BODY> Tag

- b. html tags that use a 'class' and
- c. The <DIV> tag.

3.7 CSS, Span& DIV

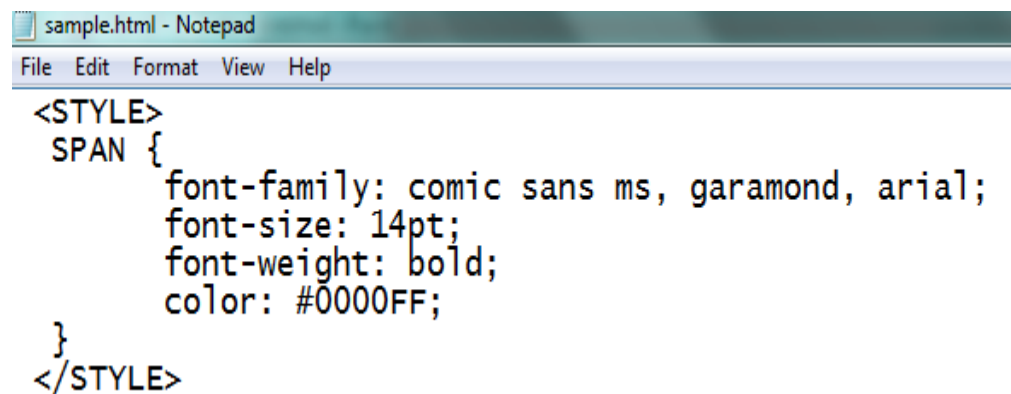
 is an inline element, which means it can start on the same line (like the font tag) and the <div> is a block level element, which means it must start on a new line (like a html table)

3.7.1 Span

The SPAN element has very similar properties to the DIV element, in that it, along with CSS, can change the style of the text it encloses. But without any style attributes, the SPAN element does not change the enclosed text at all.

 is an inline element, which means it can start on the same line (like the font tag).

An example of span...



```
sample.html - Notepad
File Edit Format View Help
<STYLE>
  SPAN {
    font-family: comic sans ms, garamond, arial;
    font-size: 14pt;
    font-weight: bold;
    color: #0000FF;
  }
</STYLE>
```

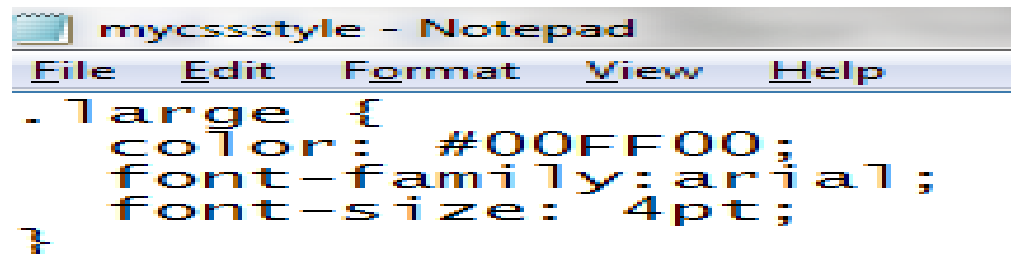
Fig. 1.13: Span Style

 this is span

3.7.2 DIV

DIV (division) element is *block-line* (which is basically equivalent to having a line-break before and after it) and used to group larger chunks of code. DIV divides the content into individual sections. Each section can then have its own formatting, as specified by the CSS. DIV is a block-level container, meaning that there is a line feed after the </div> tag.

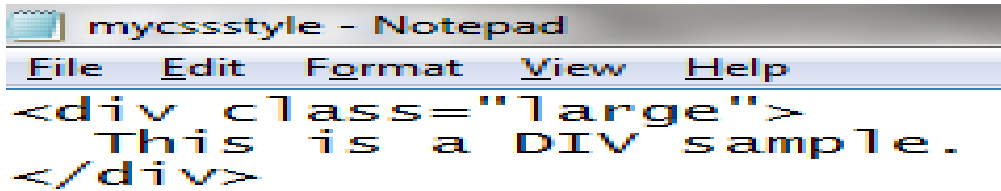
For example, if we have the following CSS declaration:



```
mycssstyle - Notepad
File Edit Format View Help
.large {
  color: #00FF00;
  font-family: arial;
  font-size: 4pt;
}
```

Fig. 1.14: Div as a Block line

The HTML code



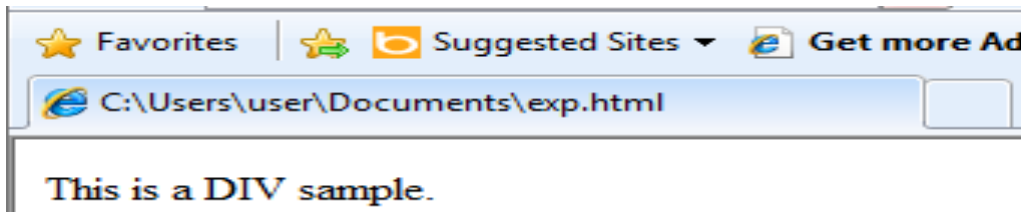
```

mycssstyle - Notepad
File Edit Format View Help
<div class="large">
  This is a DIV sample.
</div>

```

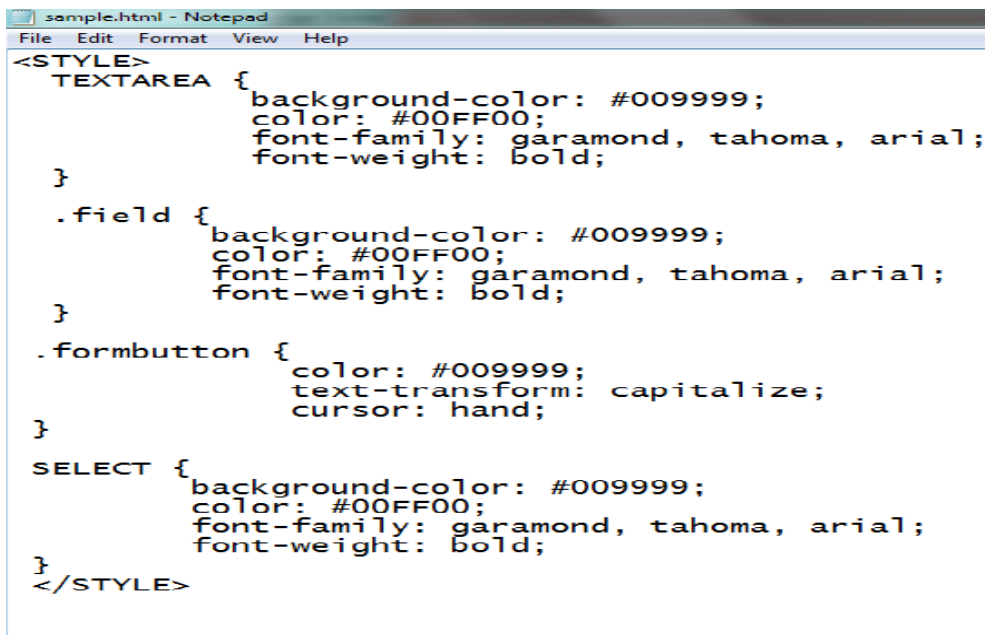
Fig. 1.15: Div Tag Appear in a Block Mode

Gets displayed as:

**Fig. 1.16: Div Output**

3.8 Cascading Style Sheets and Html Forms

With a cascading style sheet used to alter the value of any html form tag even the buttons look spectacular with style sheets. Below are the style commands that were used to achieve these effects below are the style commands that were used to achieve these effects:



```

sample.html - Notepad
File Edit Format View Help
<STYLE>
  TEXTAREA {
    background-color: #009999;
    color: #00FF00;
    font-family: garamond, tahoma, arial;
    font-weight: bold;
  }
  .field {
    background-color: #009999;
    color: #00FF00;
    font-family: garamond, tahoma, arial;
    font-weight: bold;
  }
  .formbutton {
    color: #009999;
    text-transform: capitalize;
    cursor: hand;
  }
  SELECT {
    background-color: #009999;
    color: #00FF00;
    font-family: garamond, tahoma, arial;
    font-weight: bold;
  }
</STYLE>

```


Fig. 1.17: Cascading Style Sheets and Html Forms

TEXTAREA' obviously alters the text area, 'SELECT' alters the dropdown menu and the class '.field' alters the 'text input'

```
<INPUT type=text class="field">
```

If style commands were added to the 'INPUT' tag the form buttons would also appear to have lime text on a teal background, so the class 'form button' is used.

```
<INPUT type=submit value=Send class="form button">
```

This gives the form buttons teal coloured text and capitalised the first letter of each word and for users with internet explorer the cursor changes to a hand when the mouse is over a button.

3.9 Alternative Style Sheets and Link

Firefox offers support for alternative style sheets. Pages that provide alternative style sheets allow the user to select the style in which the page is displayed using the view page style submenu. This provides a way for users to see multiple versions of a page, based on their needs or preferences. A web page can use the link element to add alternative style sheets to a document.

For example:

```
<link href="default.css" rel="stylesheet" type="text/css" title="Default
Style">
<link href="fancy.css" rel="alternate stylesheet" type="text/css"
title="Fancy">
<link href="basic.css" rel="alternate stylesheet" type="text/css"
title="Basic">
```

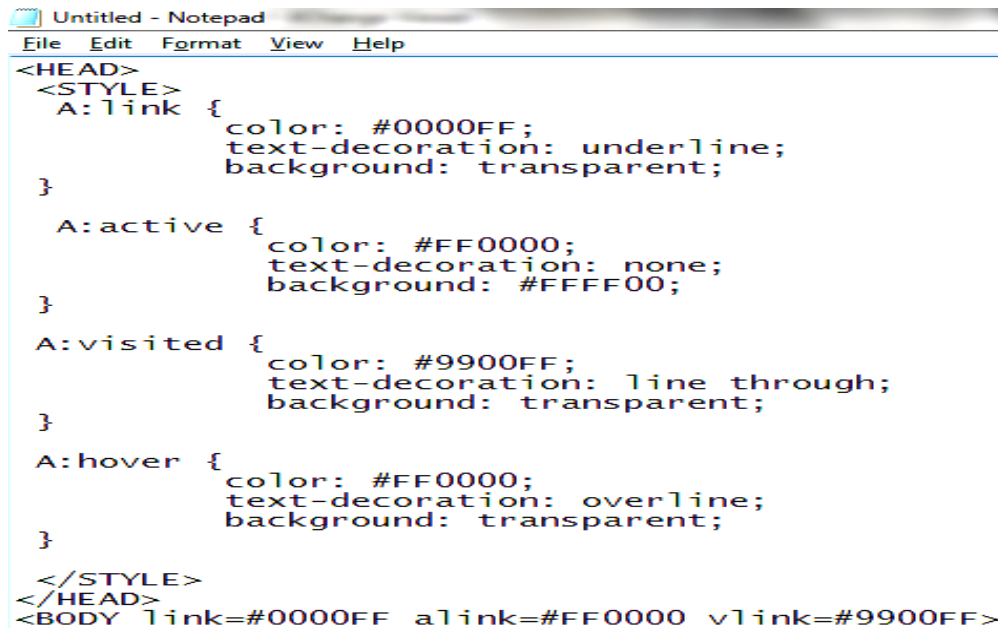
With these three style sheets offered, the styles "Default Style", "Fancy", and "Basic" will be listed in the page style submenu. When the user selects a style, the page will immediately be re-rendered using that style sheet.

When style sheets are referenced with a title attribute on the <link> or <style> element, the title becomes one of the choices offered to the user.

Style sheets linked with the same title are part of the same choice. Style sheets linked without a title attribute are always applied.

Use `rel="stylesheet"` to link to the default style, and `rel="alternate stylesheet"` to link to alternative style sheets. This tells web browser which style sheet title should be selected by default, and makes that default selection apply in browsers that do not support alternate style sheets.

It can become necessary to add alternative links in addition to normal links, for instance a document with a 'navigation panel' which has a different background colour may require links to be a lighter colour than the links in the main document, or have a completely different style, removing the underline from links etc... The example below illustrates how to set up the main document links. The link colours are still set in the body tag for the benefit of users that have browsers that are incompatible with style sheets or simply have CSS disabled, the style commands in this example describe some of the effects that can be achieved when applying style sheets to links.



```

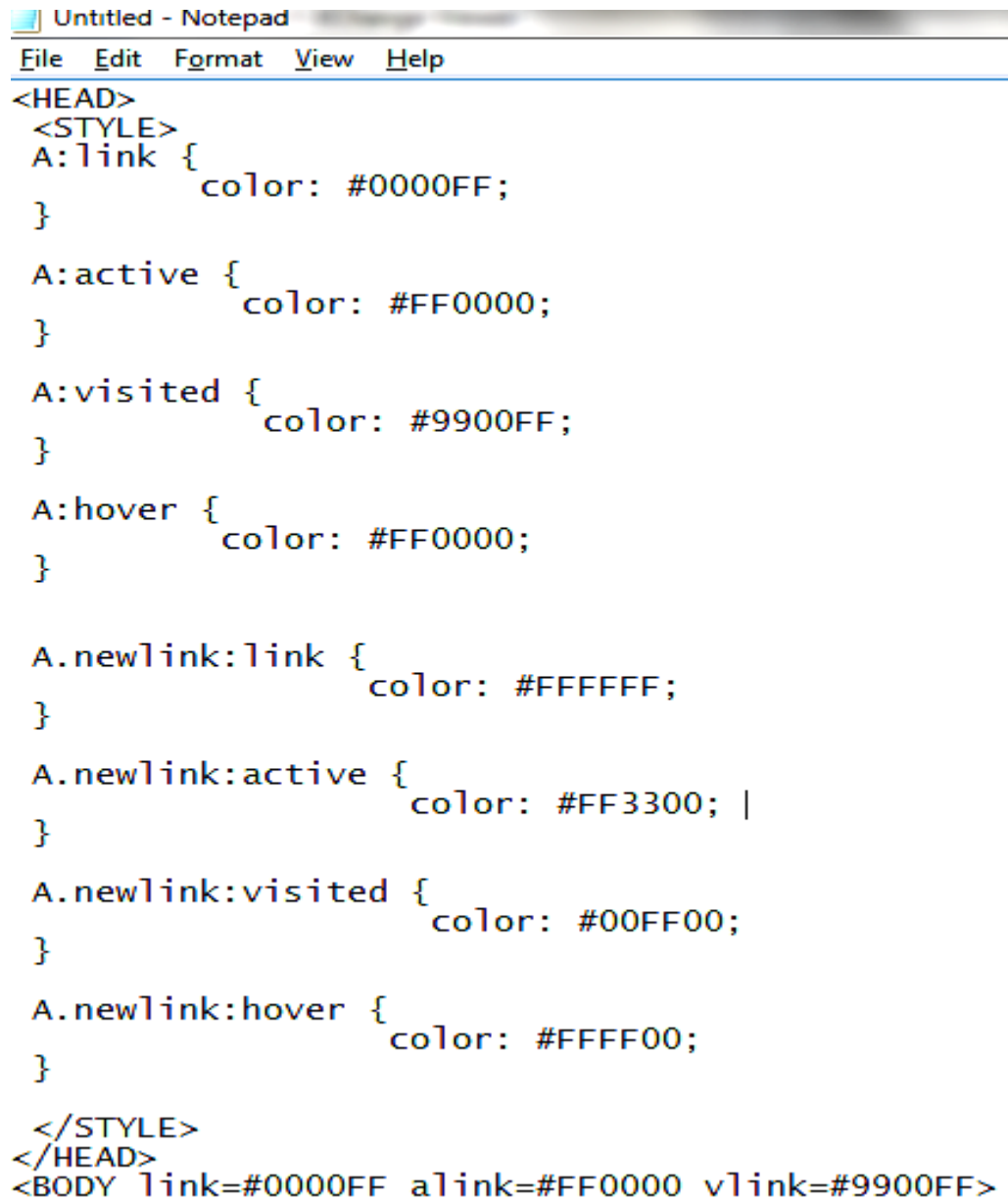
Untitled - Notepad
File Edit Format View Help
<HEAD>
<STYLE>
  A:link {
    color: #0000FF;
    text-decoration: underline;
    background: transparent;
  }
  A:active {
    color: #FF0000;
    text-decoration: none;
    background: #FFFF00;
  }
  A:visited {
    color: #9900FF;
    text-decoration: line through;
    background: transparent;
  }
  A:hover {
    color: #FF0000;
    text-decoration: overline;
    background: transparent;
  }
</STYLE>
</HEAD>
<BODY link=#0000FF alink=#FF0000 vlink=#9900FF>

```

Fig. 1.18: Alternative Style Sheets and Link

Both the body tag and the embedded style sheet will make links blue, active links red and visited links purple. The additional properties declared in the style sheet give active links a yellow background, a strong contrast to their red colour, the underline is also removed with the 'text-decoration: none;' statement. The style sheet makes visited links purple with a ~~line through~~

and links change to red in colour and are over-lined as the cursor is passed over them.



```

Untitled - Notepad
File Edit Format View Help
<HEAD>
<STYLE>
A:link {
    color: #0000FF;
}
A:active {
    color: #FF0000;
}
A:visited {
    color: #9900FF;
}
A:hover {
    color: #FF0000;
}

A.newlink:link {
    color: #FFFFFF;
}
A.newlink:active {
    color: #FF3300;
}
A.newlink:visited {
    color: #00FF00;
}
A.newlink:hover {
    color: #FFFF00;
}
</STYLE>
</HEAD>
<BODY link=#0000FF alink=#FF0000 vlink=#9900FF>

```

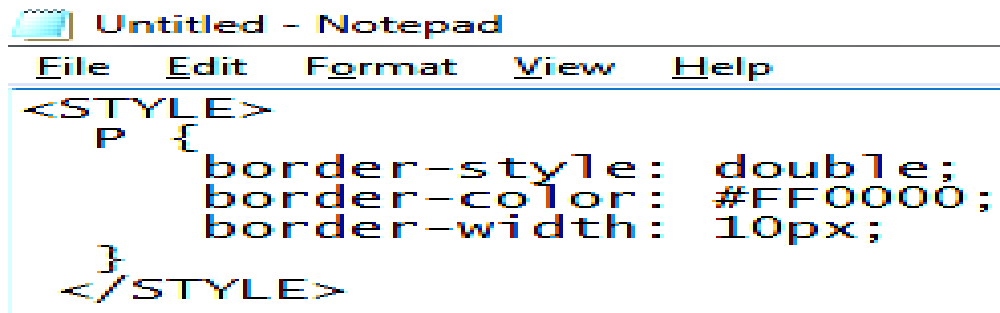
To make alternative links, a class is added to each of the link attributes.

Fig. 1.19: Alternative Style Sheets and Link, Adding Class to Link Attributes

In the example above the class 'newlink' is used to specify alternative links, for the sake of clarity only the colors have been specified but of course text-decoration, background colors or any valid CSS properties could be added

3.10 Cascading Style Sheets and Borders

This is a shorthand property which allows an author to specify 'border-top-style', 'border-right-style', 'border-bottom-style', and 'border-left-style' properties using a single property and value notation (the values are given in this order separated by spaces.) If one or more of the values are not present, the value for a missing side is taken from the opposite side that is present. If only one value is listed, it applies to all sides. This example looks at border-width, border-colour and border-style using a style sheet with the <P> tag.



```

Untitled - Notepad
File Edit Format View Help
<STYLE>
  P {
    border-style: double;
    border-color: #FF0000;
    border-width: 10px;
  }
</STYLE>

```

Fig. 1.20: Cascading Style Sheets and Borders

This is a double border, the border-width is set to 10px (10 pixels), the border-width can also be set to thin, medium or thick (border-width: medium).

Below is a simple way to write out a border for the <P> tag

```
<P STYLE="border: double #FF0000 10px">
```

Notice that just the 'border' command is used (not border-style, border-color or border-width) the examples below demonstrate more border styles

```

border-style: ridge
border-style: solid
border-style: inset
border-style: outset

```

3.10.1 The Border Commands

There are three you should be concerned with for this moment:

- Border-Width
- Border-Color
- Border-Style

Width. Color. Style. That is about all you can change. But the nice thing here is that the width, the color, and the style for a border will never be the same word. That means you will not have to write out each command each time. Just use "border:" and list the attributes. Here's an example using a block of text:

```
<P STYLE="border: double #CCFFCC 20px">
```

Here's a small block of text.

It is here to allow a border to go around it.

Blah, blah, blah. </P>

...and this is what you get:

Here's a small block of text. It is here to allow a border to go around it.
Blah, blah, blah.

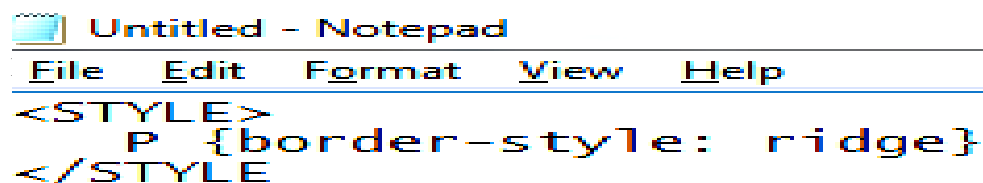
I set the "border-style" to "double". I set the "border-color" to "#CCFFCC" (please note the leading pound sign: #). Finally, I set the "border-width" to 20 pixels. The browser knows what I mean simply because none of the attributes to width, color, or style will ever be the same, so they have to represent those items by default. If you do not like to write them all out, go ahead. Just separate each with a semicolon and add one at the end. Like this below:

```
<P STYLE="border-style: dotted; border-color: #ccffcc";>
```

3.10.2 Setting the Border Commands

I have these Style Sheet commands embedded right into the element I wish to affect. I am using the STYLE="--" format to do it. If you like, you can put these into the <HEAD> section of your page in a style block. It will look something like this:

```


  A screenshot of a Notepad window titled "Untitled - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". Below the menu bar, the following HTML code is displayed:
  

```

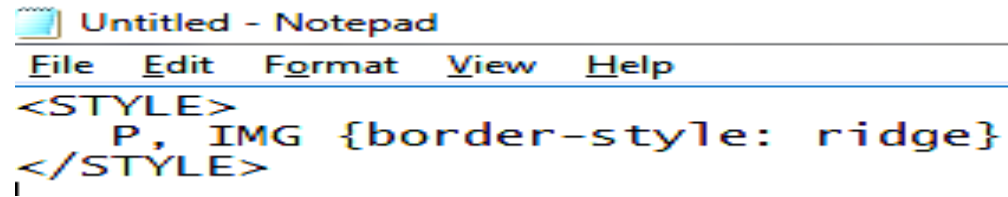
<STYLE>
 P {border-style: ridge}
</STYLE>

```


```

Fig. 1.21: Setting the Border Commands

Now every P tag on the page will get that border. If you'd like to set the same effect to multiple items, then follow this format:



```

Untitled - Notepad
File Edit Format View Help
<STYLE>
  P, IMG {border-style: ridge}
</STYLE>
  
```

Fig. 1.22: Setting the Border Commands on Multiple Item

Now all paragraphs and images will get that same border effect.

3.10.3 The Border Style Commands

I do not feel like I have to go into the "border-color" or the "border-width" command too much. Just remember that when you use the border-color, use hex codes with leading pound signs. Yes, you can also use text colour codes. No pound sign is required.

In terms of the "border-width" command, stay with pixels denoted by px, or use the "thin," "medium," or "thick" attributes. I set the color to #FF00FF so you could see the borders. They look like this:

border-width: thin

border-width: medium

border-width: thick

Now let's have some fun. There are eight different border styles available through Style Sheets: dashed (supported by MAC version browsers), dotted (supported by MAC version browsers), double, groove, inset, outset, ridge, and solid. Here's a quick look at each with the color set to #FF00FF and the border set to 10px:

border: dashed #FF00FF 10px

border: dotted #FF00FF 10px

border: double #FF00FF 10px

border: groove #FF00FF 10px

border: inset #FF00FF 10px

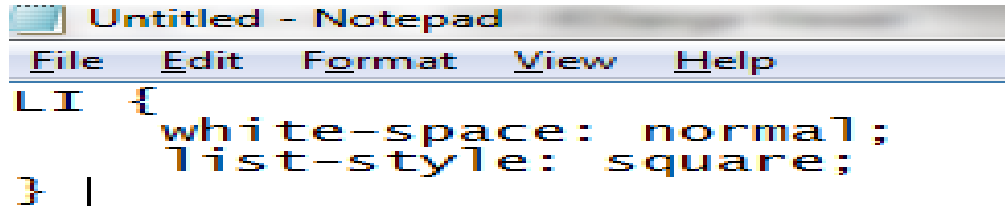
border: outset #FF00FF 10px

border: ridge #FF00FF 10px

border: solid #FF00FF 10px

3.11 Cascading Style Sheets and HTML Lists

HTML list properties can be altered in a number of ways with style sheets in the example below the value of white-space is set to normal which makes multiple spaces collapse into just one, other values that could be attributed to white-space are 'pre' which does not collapse multiple spaces and 'nowrap' which does not allow line wrapping without a
 tag.



```

LI {
    white-space: normal;
    list-style: square;
}

```

Fig. 1.23: Cascading Style Sheets and HTML Lists

The style sheet also affects the value of the list-style, giving each list item square style

- each list item
- starts with a square

Other list-styles include disc, circle, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha and none

- circle (for example; o o o o o)
- disc (for example; o o o o o)
- decimal (numbers, like an ordered list, 1 2 3 4 5)
- lower-roman (lower case roman numerals, i ii iii iv v)
- upper-roman (upper case roman numerals, I II III IV V)
- lower-alpha (lower case, a b c d e)

- upper-alpha (upper case, A B C D E)
- none, it's gone

CSS has really allowed web page designers to be freer with what they have. Below, I have tried to lay out the opportunities provided by the CSS:

- "list-style-image"
- "list-style-position"
- "list-style-type"
- and the catch-all: "list-style" commands.

I am putting this CSS command in each , , or straight away uses the format:

```
STYLE="list-style-image: something"
```

You can also use this in a style block at the beginning of the HTML document:

```
<STYLE>
  UL {list-style: circle inside lower-alpha}
</STYLE>
```

3.11.1 List-Style-Image

This command denotes an external image you'd like to use as your list item identifier. Here's what it looks like (I'll explain the two attributes right afterward):

```
<UL STYLE="list-style-image: url (green_bullet.gif)">
<LI> List Item
<LI> List Item
<LI STYLE="list-style-image: none"> List Item
</UL>
```

...and this is what you will get:

- List Item
- List Item
- List Item

This CSS command has only two attributes. You can either set up the path to the image you want to use through the url (path/image.gif) format or set the style to "none." Look at the above example again. I have set the entire UL list to use an image titled "green_bullet.gif". However, I do not want

the last list item to carry the image. That one should have the basic round bullet, so I set its list-style-image to "none".

The command is good for , , , <DD>, and <DT> tags.

3.11.2 List-Style-Position

This command allows you to denote whether the text will fall inside or outside of the bullet when it wraps. First I will allow a normal bullet, then one with the "inside" attribute, and then one with the "outside" attribute. It looks like this:

```
<UL>
<LI> this is a long list item text in order to show the text wrapping
    <LI STYLE="list-style-position: inside"> this is a long list item text
    in order to show the text wrapping
    <LI STYLE="list-style-position: outside"> this is a long list item
    text in order to show the text wrapping
```

```
</UL>
```

...and this is what you will get:

```
0    This is a long list item text in order to show the text wrapping
0    This is a long list item text in order to show the text wrapping
0    This is a long list item text in order to show the text wrapping
```

The command is good for , , , <DD>, and <DT> tags.

3.11.3 List-Style-Type

Now we get into the fun of this command. The list-style-type command allows you to set the entire list, or just one element, to a specific type.

You've done this before in HTML by adding TYPE="--" to the main or tag. Here, you're offered a little more freedom. Let's take a look at the Unordered List styles first:

```
<UL STYLE="list-style-type: square">
<LI> List Item
<LI> List Item
```

```
<LI> List Item
</UL>
```

...and this is what you will get:

- List Item
- List Item
- List Item
-

There are actually three types to choose from: circle, square, and disc. Here are all three:

```
<UL>
<LI STYLE="list-style-type: circle"> List Item
<LI STYLE="list-style-type: square"> List Item
<LI STYLE="list-style-type: disc"> List Item
</UL>
```

...and this is what you will get:

- List Item
- List Item
- List Item

But what about those ordered lists? Well, you get a few more choices there: decimal, lower-alpha, upper-alpha, lower-roman, and upper-Roman. You can pretty much guess what each looks like, but because I'm a cut-and-paste fool, here's what it all looks like:

```
<OL>
<LI STYLE="list-style-type: decimal"> List Item
<LI STYLE="list-style-type: lower-alpha"> List Item
<LI STYLE="list-style-type: upper-alpha"> List Item
<LI STYLE="list-style-type: lower-roman"> List Item
<LI STYLE="list-style-type: upper-roman"> List Item
</OL>
```

...and this is what you will get:

1. List Item
- b. List Item
- C. List Item
- iv. List Item

V. List Item

And just to drive the point home:

- decimal: basic 1, 2, 3, counting
- lower-alpha: lowercase letters
- upper-alpha: uppercase letters
- lower-roman: lowercase Roman numerals
- upper-roman: uppercase Roman numerals

The command is good for , (changes bullets to numbers/letters), , <DD>, and <DT> tags.

3.11.4 List-Style

Tired of writing all those list-item types? Do you want something that does it all for you? Well, search no more. The list-item catch-all command is for you!

You may not have noticed above, but none of the attributes across all of the list-item-something commands were equal. That means if you use "circle" or "inside" the browser knows you mean a specific list-item type. Thus, you only need this one list-item command.

Let's say I want a list with uppercase Roman numerals and wrapping text to sit outside of the Roman number. I could set both a list-style-position and a list-item-type, but why? There's no need. Use the basic list-style command and the browser will know what you mean from the attribute. Like so:

```
<OL STYLE="list-style: upper-roman outside">  
<LI> List Item  
<LI> List Item  
<LI> List Item  
</OL>
```

...and this is what you will get:

- i. List Item
- ii. List Item
- iii. List Item

4.0 CONCLUSION

Style sheets bring rich and powerful presentation techniques to the Web.

Further, the ability to separate presentation rules into a separate file simplifies HTML authoring, and makes it easy to establish and maintain a consistent presentation even across the largest web site. Here, we have shown only a few of the basic and commonly used capabilities available through style sheets.

For the CSS rules to act properly on your HTML document, it is important that the HTML and the CSS be error-free. If there are any errors in the HTML or in the CSS, the different browsers will react differently, apply the style in an unexpected way, or not apply the style at all. The online HTML and CSS validators listed in the references section make it easy to ensure that the CSS and HTML that make up your documents are error-free. We strongly encourage their use.

5.0 SUMMARY

This tutorial has taught you how to create style sheets to control the style and layout of multiple web sites at once. You have learned how to use CSS to add backgrounds, format text, add and format borders, and specify padding and margins of elements. You have also learned how to position an element, control the visibility and size of an element, set the shape of an element, place an element behind another, and to add special effects to some selectors, like links.

6.0 TUTOR-MARKED ASSIGNMENT

- i. What is CSS?
- ii. Discuss six (6) advantages of CSS
- iii. a. Write a CSS code snippet that will define the style about to apply to make changes to your HTML tags
- iv. Write a CSS code snippet to link your external style sheet to a new HTML page
- v. Where in HTML document will a link to an external style sheet code appear?
- vi. State the four main methods of applying style sheets to a html document.
- vii. Differentiate between the four main methods of applying style sheets to a HTML document using code snippet.
- viii. What are the benefits of applying a uniform look to the entire website?

7.0 REFERENCES/FURTHER READING

CSS Beginner's Guide. (n.d.).

[www.marcomm.upm.edu.my/dokumen/13032_CSS_Beginner\[1\].pdf](http://www.marcomm.upm.edu.my/dokumen/13032_CSS_Beginner[1].pdf)

CSS tutorial (2007). 'Tutorialpoint.com'.

www.tutorialspoint.com/css/index.htm.

Jen, O. M. (2007). *CSS Basics and Samples*.

<http://meiert.com/en/blog/20070922/user-agent-style-sheets/>

Mary Had a Little Lamb (2008). *Beginner CSS Tutorial*.

<http://www.gobookeee.com/css-tutorials-for-beginners/>

UNIT 3 JAVASCRIPT

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 What is JavaScript?
 - 3.2 What do you Need?
 - 3.3 Why Do I Need JavaScript?
 - 3.4 Declaring JavaScript
 - 3.5 JavaScript Syntax
 - 3.6 Displaying Information
 - 3.7 Browsing Windows
 - 3.8 Link Events, Image Swaps and the Taskbar
 - 3.9 If and Loop
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor- Marked Assignment
- 7.0 References/Further Reading

1.0 INTRODUCTION

JavaScript is a scripting language produced by netscape for use within HTML web pages. JavaScript is loosely based on Java and it is built into all the major modern browsers. This tutorial will give an initial encouragement to introduce you to JavaScript.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- define JavaScript
- differentiate between the HTML and the JavaScript tags
- differentiate between the XHTML and the JavaScript tags.

3.0 MAIN CONTENT

3.1 What is JavaScript?

JavaScript is used in designing interactive pages on a website like a calculation, pop-up-window, some web counters and even some navigation

systems. JavaScript must not be confused with Java. Java is a completely different programming language. That is used for text effects and games, although there are some JavaScript games around.

JavaScript is:

- A lightweight, interpreted programming language
- Designed for creating network-centric applications
- Complementary to and integrated with Java
- Complementary to and integrated with XHTML
- Open and cross-platform

JavaScript allows you to create ‘dynamic, interactive’ web pages: web pages that do things in response to what your readers do, without having a separate page for each possible response. In this material, I am going to cover some of the very basics of using JavaScript in your web pages. I am going to assume that you know a little bit about XHTML, but nothing about programming or JavaScript.

3.2 What Do I Need?

You will not need any special hardware or software to write JavaScript, you can just use Notepad or any other text editor. You do not even need to have any special software on your web server. JavaScript will run on any web server anywhere!

All you will need to do is make sure that you have at least a version 3 browser as versions of internet explorer, before this do not support JavaScript, so you will not be able to see your creations.

3.3 Why Do I Need JavaScript?

JavaScript can allow you to create new things on your website that are both dynamic and interactive, allowing you to do things like find out some information about a user (like monitor resolution and browser), check that forms have been filled in correctly, rotate images, make random text, do calculations and many other things.

It is necessary to already have a basic understanding of HTML before reading this study material.

3.4 Declaring JavaScript

Adding JavaScript to a web page is actually surprisingly easy! To add a JavaScript all you need to add is the following:

```
<script language="JavaScript">  
JavaScript  
</script>
```

As you can see the JavaScript is just contained in a normal HTML tag set. The next thing you must do is make sure that the older browsers ignore it. If you do not do this the code for the JavaScript will be shown which will look awful.

There are two things you must do to hide the code from older browsers and show something instead:

```
<script language="JavaScript">  
<!--Begin Hide  
JavaScript  
// End Hide-->  
</script>  
<noscript>  
XHTML Code  
</noscript>
```

As you can see this makes the code look a lot more complex, but it is really quite simple. If you look closely you can see that all that has been done is that the JavaScript is now contained in an XHTML comment tag. This is so that any old browsers which do not understand the `<script>` but will just think it is an XHTML comment and ignore it.

Because of this the `<noscript>` tag was created. This will be ignored by browsers which understand `<script>` but will be read by the older browsers. All you need to do is put between `<noscript>` and `</noscript>` what you want to appear if the browser does not support JavaScript.

This could be an alternative feature or just a message saying it is not available. You do not have to include the tag if you do not want anything shown instead.

3.5 JavaScript Syntax

A JavaScript consists of JavaScript statements that are placed within the `<script>... </script>` XHTML tags in a web page. You can place the

<script> tag containing your JavaScript anywhere within you web page but it is preferred way to keep it within the <head> tags.

The <script>tag alerts the browser program to begin interpreting all the text between these tags as a script. So simple syntax of your JavaScript will be as follows <script ...> JavaScript code </script>

The Script Tag Takes Two Important Attributes:

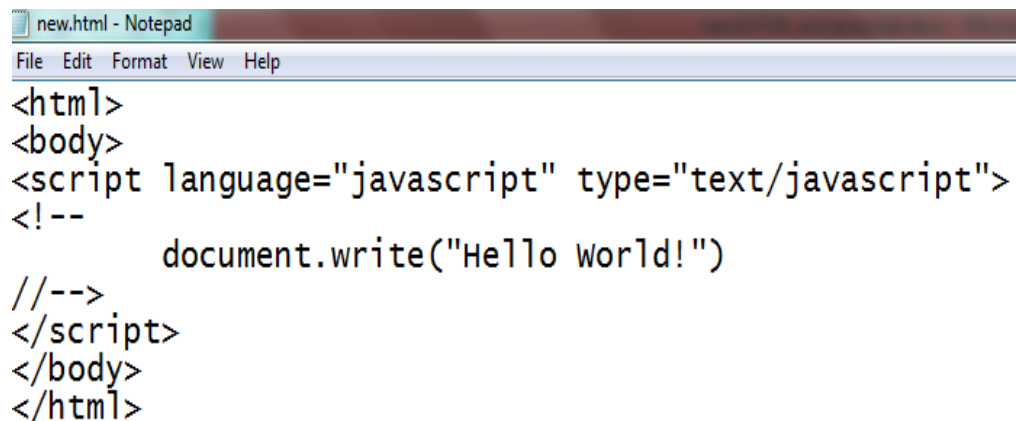
- **Language:** This attribute specifies what scripting language you are using. Typically, its value will be JavaScript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/JavaScript".

After this you will now have your JavaScript segment look like:

```
<script language="javascript" type="text/javascript"> JavaScript code
</script>
```

Your first JavaScript example:

Let us write our class example to print out "Hello World".

A screenshot of a Notepad window titled "new.html - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The text content of the window is as follows:

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
    document.write("Hello world!")
//-->
</script>
</body>
</html>
```

Fig. 2.1: JavaScript Syntax

The above code will display following result:

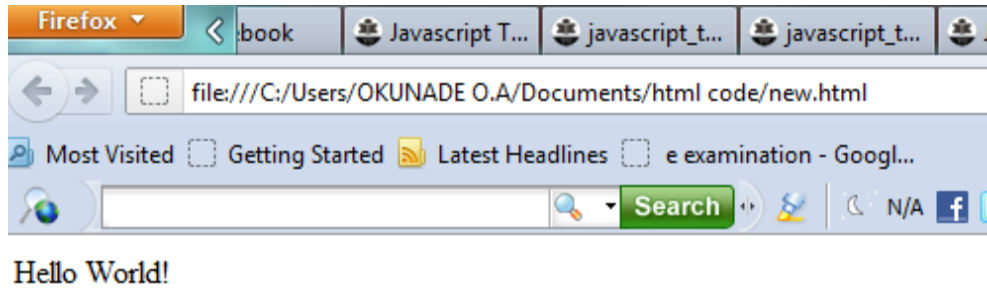


Fig. 2.2: JavaScript Syntax Output

3.5.1 Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. Because you can use spaces, tabs, and newlines freely in your program so you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

3.5.2 Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++ and Java. JavaScript, however, allows you to omit this semicolon if your statements are each placed on a separate line. For example, the following code could be written without semicolons

```
<script language="javascript" type="text/javascript">
<!--
    var1 = 10
    var2 = 20
-->
</script>
```

But when formatted in a single line as follows, the semicolons are required:

```
<script language="javascript" type="text/javascript">
<!--
    var1 = 10; var2 = 20;
-->
</script>
```

Note: It is a good programming practice to use semicolons.

3.5.3 Case Sensitivity

JavaScript is a case-sensitive language. This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalisation of letters.

So identifiers Time, Time and TIME will have different meanings in JavaScript.

Note: Care should be taken while writing your variable and function names in JavaScript.

3.5.4 Comments in JavaScript

JavaScript supports both C-style and C++-style comments, thus:

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as

3.5.6 JavaScript Data Types

JavaScript allows you to work with three primitive data types:

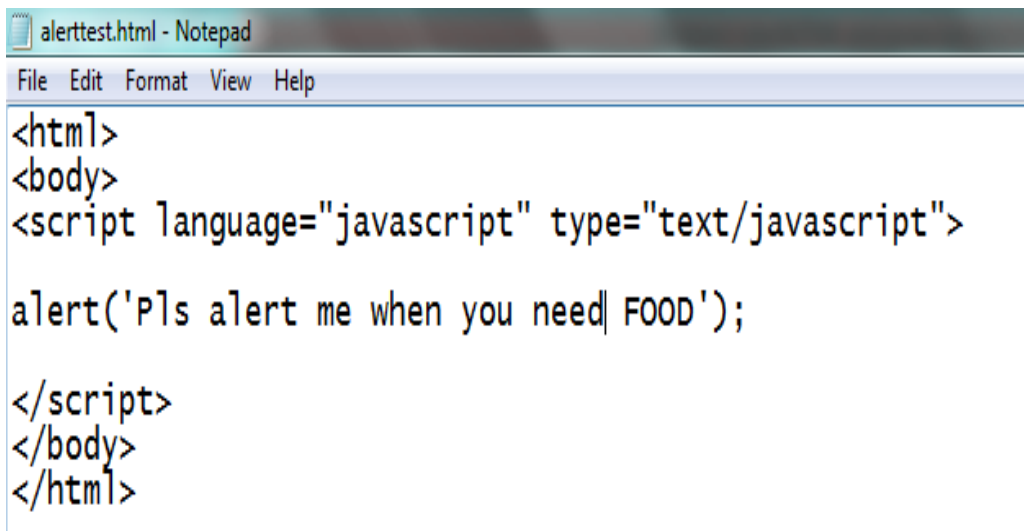
- Numbers, for example. 123, 120.50 etc.
- Strings of text, for example. "This text string" etc.
- Boolean, for example true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value.

3.5.7 Alerts

The first JavaScript command I will show you is:
alert()

This command will make a popup box appear. This can be useful for warning users about things or (in some cases) giving them instructions. It is used in the following way:
alert ('Text for alert box');

A screenshot of a Notepad window titled 'alerttest.html - Notepad'. The window has a menu bar with 'File', 'Edit', 'Format', 'View', and 'Help'. The text area contains the following HTML code:

```
<html>
<body>
<script language="javascript" type="text/javascript">
alert('Pls alert me when you need FOOD');
</script>
</body>
</html>
```

For example:

Fig. 2.3: Alert

The output looks like:

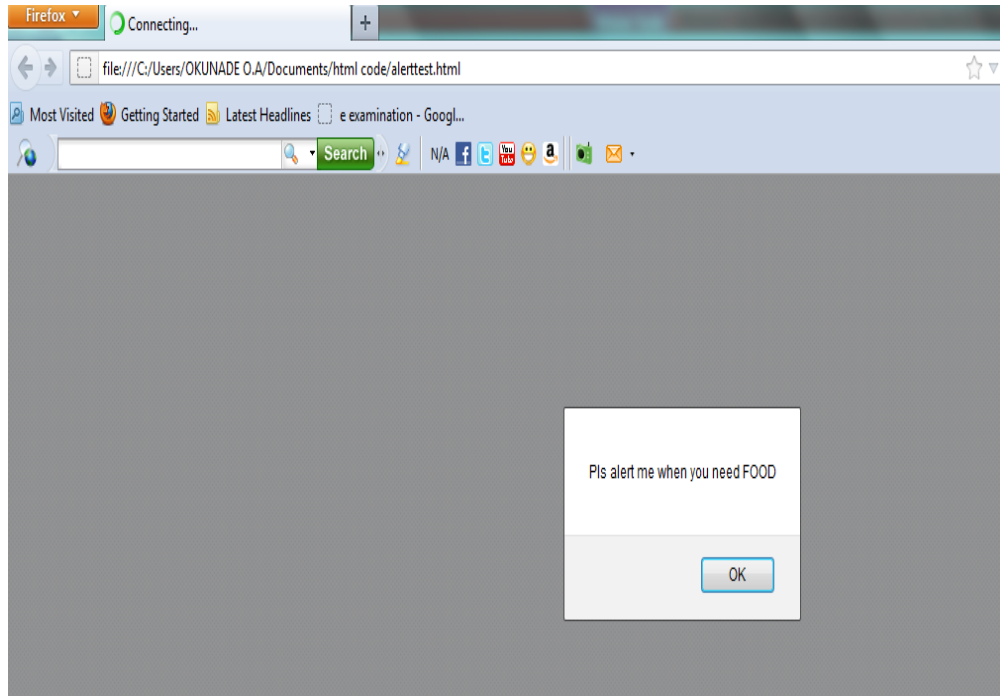


Fig. 2.4: Alert Output

In the example above I have used single quotations ' but you could use double quotations if you want to ". They work exactly the same way. The reason I use single quotes is because, later on, when you are using XHTML code and JavaScript together you will need to use them and it is good to be consistent.

Here is the full JavaScript for the earlier example:

```
alatt.html - Notepad
File Edit Format View Help
<script>
<!-- Start Hide

// Display the alert box
alert('This text is in an alert box');

// End Hide-->
</script>
```

Fig. 2.5: Full JavaScriptAlert

This is placed between the tags of the page. As you can see, I have used a comment tag as well as the alert box code. This makes your code more readable but is not essential.

See the output below:

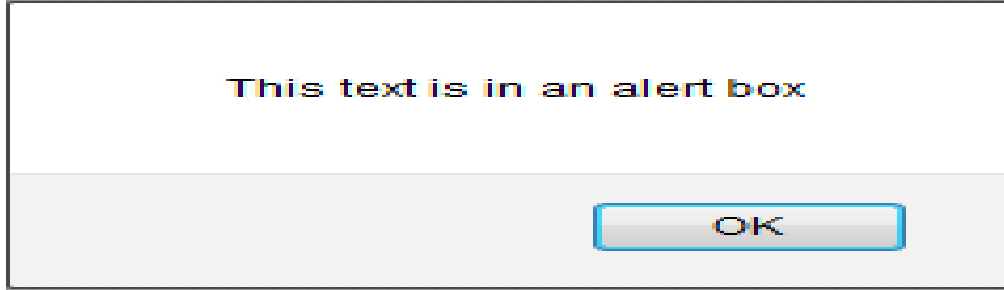


Fig. 2.6: Full JavaScript Alert Output

3.5.8 Getting Information from the User

Once you have started using variables you will realise that it will be quite useful to get some information from the user. You can do this by using the “prompt () command”.

First of all, the new prompt command is used. I set the variable “your name” using: `var your_name = prompt ('Please enter your name', 'Your Name');`

Type this on your text editor as follows:

Fig. 2.7: Getting Information from User

```
promptRequest.html - Notepad
File Edit Format View Help
<script language="javascript" type="text/javascript">
<!--Start Hide

//Getting Information From The User
var your_name = prompt('Please enter your name', 'Your Name');
// End Hide-->
</script>
```

The text between the first set of quotes is what is written on the prompt box. The text between the second set of quotes is set as the default text to

be displayed on the input section of the box where the test will be received from the users as shown below:

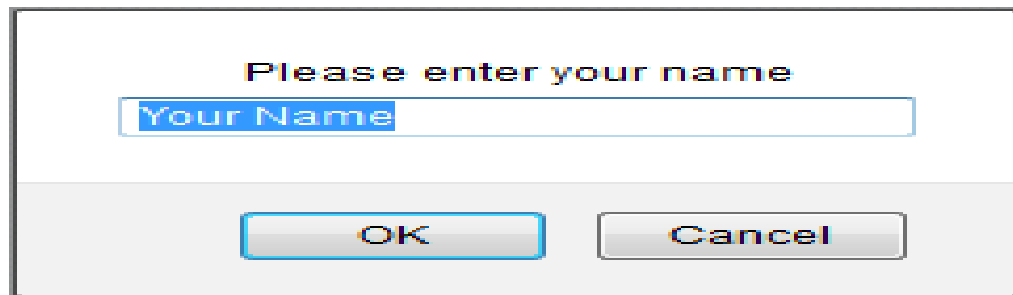


Fig. 2.8: Requesting for Information from User Displayed

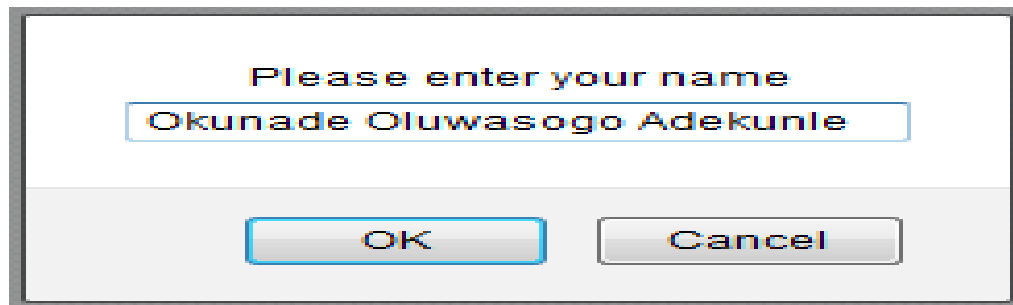


Fig. 2.9: User Input/Information Feed Back

After this I have to create the output string. I do this by adding together the input with two strings I declared earlier.

```
Varoutput_text = welcome_text + your_name + closing_text;
```

```

hi2.html - Notepad
File Edit Format View Help

<script language="JavaScript" type="text/javascript">

var your_name;
//Getting information from the user here
  your_name = prompt('Please enter your name', 'Your name');

// End Hide

//Output information here
var output_text = 'welcome'+ ' ' + your_name + ' '+to JavaScript class. Click OK to continue';
alert(output_text);

// End Hide

</script>

```

Fig. 2.10: Creation of Response to input Supplied

As you can see this is much the same as adding 3 numbers together but, as

these are strings they will be put one after the other (you could have also used quotes in here to add text and strings together). This added the text I had set as the welcome text to the input I had received and then put the closing text on the end.

Finally I displayed the output text variable in an alert box with the following code:`alert(output text);`

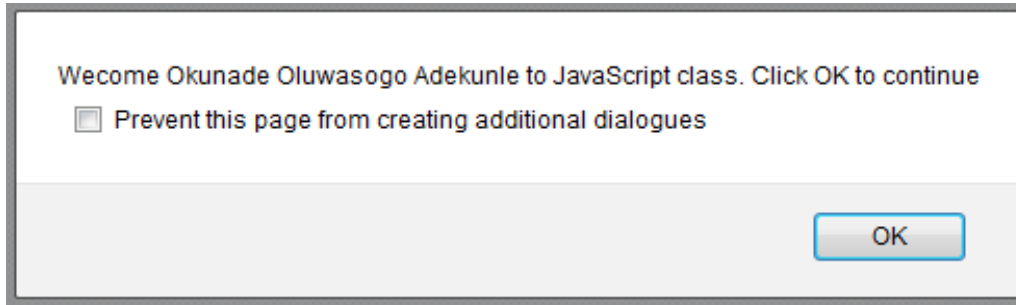


Fig. 2.11: System's Response to Input/Information Supplied Display

This, instead of having text defined as the content for the alert box, places the string in the box.

3.5.9 JavaScript Variables

Variables in JavaScript, as in other computer languages, can be used to store information. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the `var` keyword as follows:

```

variab.html - Notepad
File Edit Format View Help
<script type="text/javascript">
  <!--
  var money; var name;
  //-->
</script>

```

Fig. 2.12: JavaScript Variables

3.5.10 JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

1. **Global Variables:** A global variable has global scope which means it is defined everywhere in your JavaScript code.
2. **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

1. Numbers for example the variable: my number should have the value: 3456
2. Variables can also contain text for example the variable: my name could have the value: Okunade Adekunle

Variables can be very useful for text or numbers that you repeat several times in a program, for doing calculations or for getting input from a user. Variables are declared as follows:

1. Number: `varmy_number = 3456;`
2. Strings (text): `varmy_name = Okunade Adekunle';`

As you can see a string is included in quotes (either single or double) but a number does not. If you include a number in quotes it will not be treated as a number. You may also notice that each line ends with a semicolon. This is standard JavaScript code to show the end of a line. But it is optional.

When naming your strings you can use any word or combination of words as long as it is not already in use by JavaScript (so don't use alert as a variable name etc.). Do not include spaces, replace them with `_`.

3.5.11 JavaScript Variable Names

While naming your variables in JavaScript keep following rules in mind.

- You should not use any of the JavaScript reserved keyword as variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or the underscore character. For

example, 123test is an invalid variable name but _123test is a valid one.

- JavaScript variable names are case sensitive. For example, Name and name are two different variables.

3.5.12 JavaScript Reserved Words

The following are reserved words in JavaScript. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names:

Abstract, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double, else, enum, export, extends, false, final, finally, float, for, function, go, to, if, implements, import, in, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, type of, var, void, volatile, while, with.

3.5.13 The Arithmetic Operators

The following arithmetic operators are supported by JavaScript language:
Assume variable A holds 10 and variable B holds 20 then:

Table 2.1: Arithmetic Operators

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by the denominator	B / A will give 2
%	Modulus operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A--will give 9

3.5.14 The Comparison Operators

There are following comparison operators supported by JavaScript language.

Assume variable A holds 10 and variable B holds 20 then:

Table 2.2: Comparison Operators

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

3.5.15 The Logical Operators

There are following logical operators supported by JavaScript language. Assume variable A holds 10 and variable B holds 20 then:

Table 2.3: Logical Operators

Operator	Description	Example
&&	Called logical and operator. If both the operands are non zero then condition becomes true.	(A && B) is true.
	Called logical or operator. If any of the two operands are non zero then condition becomes true.	(A B) is true.
!	Called logical not operator. Use to reverses the logical state of its operand. If a condition is true then logical not operator will make false.	!(A && B) is false.

3.5.16 The Bitwise Operators

There are following bitwise operators supported by JavaScript language
Assume variable A holds 2 and variable B holds 3 then:

Table 2.4: Bitwise Operators

Operator	Description	Example
&	Called bitwise and operator. It performs a boolean and operation on each bit of its integer arguments.	(A & B) is 2.
	Called bitwise or operator. It performs a boolean or operation on each bit of its integer arguments.	(A B) is 3.
^	Called bitwise or operator. It performs a boolean exclusive or operation on each bit of its integer arguments. Exclusive or means that either operand one is true or operand two	(A ^ B) is 1.

is true, but not both.

- ~ Called bitwise not operator. It is a unary operator and operates by reversing all bits in the operand. ($\sim B$) is -4.
- << Called bitwise shift left operator. It moves all bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. shifting a value left by one position is equivalent to multiplying by 2, shifting two positions is equivalent to multiplying by 4, etc. (A << 1) is 4.
- >> Called bitwise shift right with sign operator. It moves all bits in its first operand to the right by the number of places specified in the second operand. The bits filled in on the left depend on the sign bit of the original operand, in order to preserve the sign of the result. if the first operand is positive, the result has zeros placed in the high Bits; if the first operand is negative, the result has ones placed in the high bits. shifting a value right one place is equivalent to dividing by 2 (discarding the remainder), shifting right two places is Equivalent to integer division by 4, and soon. (A >> 1) is 1.

>>> Called bitwise shift right (A >>> 1) is 1.
with zero operators. This
operator is just like the >>
operator, except that the
bits shifted in on the left
are always zero,

3.5.17 The Assignment Operators

The basic assignment operator is equal (=), which assigns the value of its right operand to its left operand. That is, $x = y$ assigns the value of y to x . The other assignment operators are usually shorthand for standard operations, as shown in the following table:

Table 2.5: Assignment Operators

Shorthand operator	Meaning
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x \% = y$	$x = x \% y$
$x \ll = y$	$x = x \ll y$
$x \gg = y$	$x = x \gg y$
$x \gg \gg = y$	$x = x \gg \gg y$
$x \& = y$	$x = x \& y$
$x \wedge = y$	$x = x \wedge y$
$x = y$	$x = x y$

3.5.18 The Conditional Operator (? :)

There is an operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax:

(Boolean_Condition) ? Code_for_true: Code_for_false

Code_for_true is executed if boolean_condition evaluates to true. Otherwise Code_for_false is executed.

Table 2.6: Conditional Operator

Operator	Description	Example
?:	Conditional Expression	If Condition is true? Then value X : Otherwise value Y

For example:

```
var oranges=20;
```

```
(oranges<20)? alert("Less than 20") : alert("Not less than 20");
```

The output of the above code is:

**Fig. 2.13 Conditional Operator Display/Output**

3.5.19 The *Type of Operator*

The type of is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The type of operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is the list of return values for the type of Operator:

Table 2.6: Type of Operator

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"

Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

3.5.20 Calculations

You can do calculations if you have variables containing numbers. Here is an example of some code which does a calculation:

```
// Set Variables varfirst_number = 15;
varsecond_number = 5;
varthird_number = 10;
varfourth_number = 25;

//Do Calculations
varnew_number = first_number + second number
var answer = new_number * third_number
var answer = answer / fourth_number
```

This code sets four number variables. It then adds the first and second numbers together and stores the answer as a variable called new answer. Then it multiplies new number by the third number and stores the answer as answer. Finally, it divides the answer by the fourth number to get a new value for the answer.

3.6 Displaying Information

How to display alert boxes, how to get information from the user and how variables work has been explained earlier on.

3.6.1 Document Writeln

This command is very useful as it will output information to a web page. I will start with a basic example of how to use it:

```
document.writeln('Hello there!');
```

This basically tells the JavaScript to write to the document (web page) on a new line the text Hello there!. The really useful bit of this is that you can tell the JavaScript to output text, HTML and variables. First of all I will show you how to output HTML:

```
document.writeln('<font face="Arial" size="5" color="red">Hello  
there!</font>');
```

This will display the following on the page:

Hello there!

As you can see, this allows you to just put standard HTML code into a web page using JavaScript. If you cannot see a good reason for this it is not surprising at the moment but next I will introduce variables to the script.

You can also use document.write to print variables. Enter the variable name without quotes, like so:

```
var my text = "Hello again";  
document.write(my text);
```

Note that if quote marks are placed around the variable name, the variable name itself will be printed (instead of the variable value). You can also combine variable values and text strings by using the + sign:

```
var colour1 = "purple";  
var colour2 = "pink";  
document.write ('<p>colour1: ' + colour1 + '<br>colour2: ' + colour2  
+ '</p>');
```

Notice the way the variable names are used literally as well as for passing the values. This will print the following:

```
colour1: purple  
colour2: pink
```

Remember, text inside quotes will be printed literally, text with no quotes is assumed to be a variable.

Up until now all the JavaScript code has been contained inside the tags of the HTML document. The reason for this is that the JavaScript will be loaded and executed before the rest of the document. This works fine for most scripts but, as the scripts become more advanced, you may need to have them both in the document and the head. I will use this script to demonstrate. To put JavaScript in the section of the page is exactly the same as putting it in the section of the page. I would suggest that you adopt the following way of creating scripts: Put all your initialization code in the head of the page (like setting variables, prompts, alerts etc.) and then all display code (document. writeln etc.) in the body of the page. This is the code for the new improved version of the example which uses document. writeln:

```
<body>
<script type="text/JavaScript">
<!--
var linebreak = "<br />"
var my_var = "Hello World!"

document.write(my_var)
document.write(linebreak)
my_var = "I am learning JavaScript!"
document.write(my_var)
document.write(linebreak)

my_var = "Script is Finishing up..."
document.write(my_var)
//-->
</script>
</body>
```

3.6.2 Remote JavaScript

Now you have learnt how to use the document. writeln command you can now start using one of the most useful applications of JavaScript, remote scripting. This allows you to a JavaScript in a file which can then be 'called' from any other page on the web.

This can be used for things on your own site which you may like to update site-wide (like a footer on the bottom of every page) or something used on remote sites (for example newsfeed or some counters). To insert a remote JavaScript you do the following:

```
<script type="text/javascript" src="http://www.yourdomain.com/javascript.js">
</script>
```

Which will then run the JavaScript stored at `http://www.yourdomain.com/javascript.js`. The .js file is also very simple to make, all you have to do is write your JavaScript (omitting the `<script>` tags into a simple text file and save it as something.js.

If you want to include information for browsers which do not support JavaScript you will need to put the tags in the HTML, not the JavaScript file.

There is one problem with using remote JavaScript which is that only the recent browsers support it. Some browsers which support JavaScript do not support Remote JavaScript. This makes it advisable not to use this for navigation bars and important parts of your site.

3.7 Browser Windows

In this part, you will learn how to create and manipulate browser windows in Javascript.

3.7.1 Opening a window with JavaScript

The JavaScript command used to open a window is:

3.7.1.1 window.open

For this to work, though, it requires two extra things. Firstly you will need to have some extra information so that the JavaScript knows how to open the `window:window.open('link.html','mywindow');`

This means that a window called 'mywindow' will open with the page link.html in it, exactly like with the HTML code above.

This code can either be used as part of your JavaScript code (for example if you included it in the JavaScript code in the section of your page it would

open when the pageloaded) or can be used when a link is clicked. To do this you must use another JavaScript command called onclick.

3.7.1.2 Manipulating the Window

The main reason of using JavaScript to manipulate windows, though, is because you can set many things on the window which you could never do with HTML. JavaScript allows you to use commands to decide which parts of the browser window appear.

This is done using a third part of the window.open command. This is where you decide the window features: `window.open('link.html','mywindow','window features');`

There are many things you can include here. For example if you wanted a window that only has a location bar and a status bar (the part at the bottom of the browser) then you would use the following code:

```
window.open('link.html','mywindow','location, status');
```

There are many different things you can include in your new window:

Table 2.7: Feature and Function

Feature	Function
Menu bar	The File, Edit etc. menus at the top of the browser will be shown
Scrollbar	This will show scrollbars at the side of the window if they are needed
Width	You can set the width of the window in pixels (see the next section)
Height	You can set the height of the window in pixels (see the next section)
Toolbar	This will display the browser toolbar with the Back, Stop, Refresh buttons etc.
Location	The location bar (where you type in URLs) will be displayed in the browser
Resizable	This will allow the window to be resized by the user

directories	This will show the directories in netscape browsers like 'Whats new' and 'Whats cool'
-------------	---

Examples of Manipulating Windows

You may be a little confused by all these options so I will show you a few examples of opening a window in JavaScript:

This window will open with a location bar, toolbar and will be resizable:
`window.open ('window1.htm','the_first_window','location, toolbar, resizable');`

This will open another page in this window:
`window.open ('window2.htm','thefirstwindow');`

This will open a window 200 pixels wide and 300 pixels high. It is not resizable and has a status bar and will scroll if necessary. This is a very commonly used combination:

`window.open('window1.htm','thesecondwindow','height=300,width=200,status,scrollbars');`

3.8 Link Events, Image Swaps and the Taskbar

This tutorial will show you how to use link events; do image swaps and display things in the browser status bar.

3.8.1 Link Events

A link event is a different way of including JavaScript on your page.

There are three ways of executing some JavaScript code in a link. They are:

- On Click
- On MouseOver
- On MouseOut

They can have many different uses but the most common is for image swaps (mouseover images).

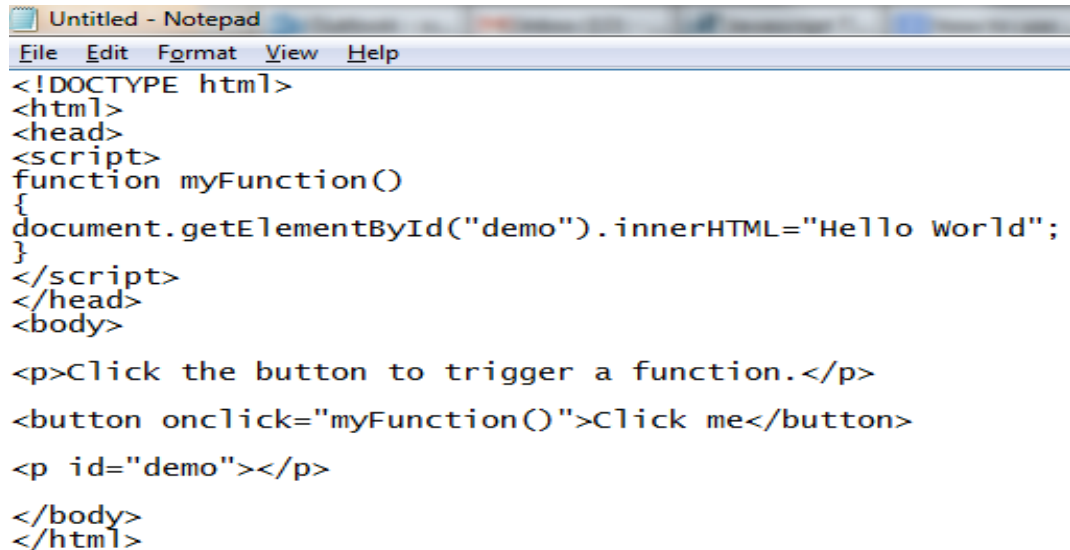
3.8.1.1 On Click

OnClick works in exactly the same way as a standard HTML link. The onclick event occurs when the user clicks on an element.

Syntax:

It is inserted as follows:

```
object.onclick=function(){SomeJavaScriptCode};
```



```

Untitled - Notepad
File Edit Format View Help
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML="Hello world";
}
</script>
</head>
<body>

<p>Click the button to trigger a function.</p>
<button onclick="myFunction()">Click me</button>
<p id="demo"></p>

</body>
</html>

```

Fig. 2.14 **onClick**

Result:

Click the button to trigger a function.

Hello World

```
<button onclick="myFunction()">Click me</button>
```

As you can see, one main difference is that the href of the link points to a #. This is nothing to do with the JavaScript, it just means that, instead of opening a new page, the link will not do anything. You could, of course, include a link in here and you would be able to open a new page AND execute some code at the same time. This can be useful if you want to change the content of more than one browser window or frame at the same time.

3.8.1.2 onmouseover and onMouseOut

onmouseover and onMouseOut work in exactly the same way as onClick

except for one major difference with each.

onMouseOver, as the name suggests, will execute the code when the mouse passes over the link. The onMouseOut will execute a piece of code when the mouse moves away from the link. They are used in exactly the same way as onClick.

3.8.2 Rollover Images

This is one of the main ways of using link events. If you have not seen rollover images before, they are images (usually used on navigation bars) and when the mouse moves over the link it changes the image which is displayed.

This is done using a combination of the onMouseOver and onMouseOut events. The image change when the mouse passes over the link you want the image to change to the new image, when it moves away from the link, the old picture should be displayed again.

The first thing you must do is edit the tag you use to insert the image you want to change. Instead of just having something like this:
 <imgsrc="home.gif" alt="Home">you would have the following:
 <imgsrc="home.gif" alt="Home" name="home_button">

The name for the image could be anything and, like the window names from the last part, is used to refer to the image later on.

Now you have given a name to the image you are using you will want to create the rollover. The first thing you must do is create the image for the rollover and save it. Then create a link round the image. If you do not want to have a link on the image you can still do a rollover by specifying the href as # which will make the link do nothing eg:

The following code will make a mouseover on your image (assuming you inserted the image as shown above and called your mouseover image homeon.gif):

```
<a href="index.htm" onMouseOver="home_button.src='homeon.gif';"
onMouseOut="home_button.src='home.gif';"><imgsrc="home.gif"
alt="Home" name="home_button" border="0"></a>See the above code's
explanation:
```


Firstly you are creating a standard link to index.htm. Then you are telling the browser that when the mouse moves over the link the image with the name home_button will change to homeon.gif. Then you are telling it that when the mouse moves away from the link to change the image called home_button to home.gif. Finally you are inserting an image called home_button with an alt tag of 'Home' and no border round it.

3.8 Status Bar

The status bar is the grey bar along the bottom of a web browser where information like, how much the page has loaded and the URL which a link is pointing to appears. You can make your own text appear in the status bar using the JavaScript you already know.

You can include this in standard code, onClick, onMouseOver or onMouseOut. This allows you to do things like display a description of a link when the mouse moves over it.

3.9 If Statements and Loops

In the previous lecture I have shown you how to declare a JavaScript, open windows and display information. In this part I will show you how to use two of JavaScripts most important functions, If and Loops.

3.9.1 If Statements

The 'if' function allows you to check to see if something is true or false. For example you could check to see if text entered by a user matches a piece of text you already have (like a password).

This could be very useful for many sites. The code is as follows:

```
var guess = prompt("Please guess a number between 1 and 10","");
if(guess == 5)
```

```
{
  alert('Correct! I was thinking of 5');
}
else
{
  alert('Wrong! I was thinking of 5');
}
```

This code is made up of three main parts. The first part which gets the guess from the user, you have used before. This is followed by:

```
if(guess==5)
```

This checks to see if the variable guess is equal to 5. The if statement is followed by:

```
{alert('Correct! I was thinking of 5');
}
```

The important part of this is the curly brackets round the alert command.

These contain the code which should be executed if the if statement returns 'true' (in this example if guess equals 5). The final part is the: else

```
{alert('Wrong! I was thinking of 5');
}
```

This tells the browser that if the if statement does not return 'true' (in this example if guess does not equal 5) to execute the code between the curly brackets.

Together this code makes up the 'if' statement.

3.9.2 More 'If' Statements

There are other conditions you can test with the if statement. Firstly, as well as using numbers you can compare variables or text:

```
if(variable == variable)
```

```
if(variable == "some text")
```

As well as doing this you can check to see if one variable is greater than another, less than another or not the same as another:

```
if(variable < other variable)
```

If variable is less than other variable

```
if(variable > other variable)
```

If variable is greater than other variable

```
if(variable <= other variable)
```

If variable is less than or equal to other variable

```
if(variable >= other variable)
```

If variable is greater than or equal to other variable

```
if(variable != other variable)
```

If variable is not equal to other variable

These can be very useful in your web pages. You can also check to see if two conditions are true before executing code using `&&`:

```
if(variable1 == variable2) && (variable1 < variable3)
```

This will only execute its code if variable1 is equal to variable2 and is less than variable3. You can also run the code if one of the conditions is true:

```
if(variable1 == variable2) || (variable1 < variable3)
```

This will only execute its code if variable 1 is equal to variable 2 or is less than variable3.

3.9.3 While Loops

While loops make a pieces of code to repeat until the condition is met.

This is very useful for things like passwords when you want to keep asking the user until they get it correct. For example this code will repeat until the user enters the word 'hello':

```
var password = 'hello';  
var input = prompt('Please enter the password', "");  
while(input != password)  
{  
  var input= prompt('Please enter the password");  
}
```

This will continuously loop the code inside the `{ }` until the test input does not equal password is false (the password is correct).

3.9.4 For Loops

For loops are used to do something a set number of times. For example:

```
for(loop=0; loop < 11; loop++)  
  
{  
document.writeln(loop);  
}
```

This will start by setting the variable loop to 0, it will then loop, adding one to the value each time (using the loop++ code) as long as loop is less than 11. They take the form:for(starting value; test; increment)

3.10 Forms and Functions

Here I will show you how you can manipulate HTML forms with JavaScript to improve your website.

3.10.1 Changing the Value of a Text Box

Before you can manipulate text boxes you must first know how to create a form and how to create a text box in HTML. I will quickly explain this.

Forms for JavaScript are slightly different to standard ones as they do not require any information or script to handle them. You can do everything by just giving the form a name (although later you may want to add the other information so that you can also have the form submitted):

```
<form name="formname">  
</form>
```

In this form you can place a text box using:

```
<input type="text" name="boxname">
```

Once you have this you can create your first JavaScript to refer to a form:

Move the mouse over here to add some text to the box

This is done very easily using the onMouseOver property of the link. It can refer to the text box as follows:

```
window.document.example1.first_text.value='Hi there';
```

This tells the browser to put 'Hi there!' into the value of the item called 'first text' in the form called 'example1'. You can also do this with multiline text boxes. You can use `\n` to start a new line.

3.10.2 Events

Just like links, you can set events for items in a form. They are:

`onBlur` - Happens when the cursor is moved out of the field
`onFocus` - Happens when the cursor is moved into the field
`onChange` - Happens when the field is changed and the cursor moves out of it. These are placed into the code as part of the form's item for example:

```
<input type="text" onBlur="dothis">
```

3.10.3 JavaScript Functions

JavaScript functions are very useful, and this seems an appropriate place to introduce them. They are pieces of JavaScript which can be declared at the top of your page and can then be referred to again and again in your code.

You can even pass parameters to them and make them do different things.

Functions take the following form:

```
function function name(parameters)
{
}
```

For a very basic function you need no parameters and it would be constructed like this (in the `<head>` of the document):

```
function sayhi()
{
  alert(Hi there!);
}
```

Then, anywhere in the text you could place this:

```
<a href="#" onMouseOver="sayhi();">Say Hi</a>
```

Which would display the alert whenever the mouse passed over it. Unless you are repeating something many times, though, this will not seem particularly useful. This is where parameters become useful.

3.10.4 Parameters

Parameters are a very useful part of functions. They allow you to pass data to the function when it is called. Here is an example:

```
function say(what){alert(what);
}
in the head, then:<a href="#" onMouseOver="say(hi);">Say Hi</a>
<a href="#" onMouseOver="say(bye);">Say Bye</a>
```

What this is doing is the information in the brackets is passed to the function. In the brackets of the function is the word 'what'. This is telling the JavaScript to assign the information in the brackets to the variable what (the same as var what='something;'). You can even do this with multiple pieces of information by separating them by commas.

Functions and Parameters are very useful.

3.10.5 Using the Submit Button

The most common use of forms is to submit data to a script so that it can be processed. This is fine if you are using CGI to do a mailto form or something like that, but if you just want to run a piece of JavaScript from a form you will need to do things a little differently. The first part you must learn about is the return false; attribute. It can be used as follows:

```
<form name="myform" onSubmit="return false;">
<input type="submit" value="Submit">
</form>
```

What this code does is tell the JavaScript that when the form is submitted it should do nothing. This stops Netscape from refreshing the page (as it would do if there was just a submit button and the form had no action). Now what you can do is to call a function using the submit button:

```
<form name="myform" onSubmit="MyFunction(); return false;">
```

Which will tell the browser to run the function 'MyFunction' when the Submit button is clicked. You can, of course, use this without the: return

false;part. This would be useful if, for example, you want to validate a form before it is sent to a CGI script. You would include the form's action as normal. Then, in the onSubmit part you would call your function to check what had been entered.

3.10.6 Checkboxes

Checkboxes and radio buttons are two of the remaining form items. First I will cover checkboxes.

Checkboxes have two possible values:

- Unchecked Box
- Checked Box

As there are only two possible values, the JavaScript to refer to a checkbox is slightly different to that of a text box. In the example below the checkbox has been given the name my checkbox and is in the form example1 above.

```
if(window.document.example1.my_checkbox.checked=true)
{
alert("The box is checked!")
}
else
{
window.document.example1.my_checkbox.checked=true;
alert("The box was not checked so I have checked it!");
}
```

I will now explain what this code does. It will check the value of the checkbox. If the value is true (which means the checkbox is checked) then it will display an alert box which will tell you the box is checked. If it is not checked it will check the box (put a checkmark in it) and tell you what it has done. If you want to check or refer to the unchecked value of a checkbox the code would be:

```
window.document.example1.my_checkbox.checked=false;
```

Remember that, when referring to check boxes you do not need to enclose true or false in quotations also as with all other form objects you can use onBlur, onChange and onFocus.

3.10.4 Radio Buttons

Radio buttons are usually found in groups like this:

- Blue
- Red
- Green

Only one button in the group can be checked at a time. Radio buttons work in exactly the same way as a checkbox, having a checked property which can either be true or false.

3.10.7 Selects and Menus

Menus are drop down boxes with several options in them (they are mainly used for things like selecting your country on a form). Selects are really just menus with multiple lines and scrollbars. To show how they work I will show you a quick example of a menu in action and the code for it. For a select box I would just change the height property.

The code used is below:

```
<form name="go" onSubmit="return false;">
<select name="select"
onChange="window.document.location=window.document.go.select.value;
">
<option value="#" selected>Please Select</option>
<option value="http://www.microsoft.com">Microsoft</option>
<option value="http://www.yahoo.com">Yahoo!</option>
<option value="http://www.gowansnet.com">Gowansnet</option>
</select>
</form>
```

What this code is doing is, when the select is changed it is telling the page in the browser to change its location to the value of the select box. The location of the document is accessed using:

```
window.document.location
```


As you can see, this could be very useful, both for getting feedback from visitors and for redirecting them (as in the example above).

4.0 CONCLUSION

Now you have learnt all about the basics of JavaScript. I covered some of the most important parts of JavaScript and now you should be able to start writing some quite advanced scripts. There is still a lot of JavaScript left to learn with many more advanced commands. One way of improving your skills, though, is to look at the code of other people's pages that use JavaScript. You can learn more about the language by doing this.

5.0 SUMMARY

This unit has demonstrated what JavaScript is and how it works. We started with the basics like how to turn JavaScript on and off in your browser. We then quickly moved on to the basic syntax of JavaScript.

We learned that you can include JavaScript within your HTML page, or you can place it in an external file - the choice is yours.

We learnt about some of the key concepts of JavaScript and that these concepts are pretty standard across most programming/scripting languages concepts such as operators, variables and functions. We learned that intrinsic events enable you run a script in response to a user's action.

We covered some more of the standard programming concepts such as if statements, and loops. We learned how to catch an error and present a friendly message back to the user (so as to not scare them off).

6.0 TUTOR- MARKED ASSIGNMENT

- i. What is JavaScript?
- ii. State two uses of JavaScript
- iii. Differentiate between JavaScript and Java
- iv. Why do I need JavaScript?
- v. Write a JavaScript declaration code snippet
- vi. Write a code snippet to hide new JavaScript codes from the old browses
- vii. Write short note on the importance of <noscript> tag
- viii. Explain any 2 important attribute of a script tag

- ix. Discuss Remote JavaScript
- x. Differentiate between onMouse, onMouseOver and onMouseOut using JavaScript code to back up your point:
- xi. Write a JavaScript code to implement a rollover
- xii. Write a JavaScript code to differentiate between any two types of loop
- xiii. Show with example the different ways of using if statement
- xiv. Write a JavaScript code to differentiate between onBlur, onFocus and onChange
- xv. Write the syntax for Function JavaScript
- xvi. Write the syntax for Form JavaScript
- xvii. Write a JavaScript Function that could display an alert whenever the mouse passed over a text area.
- xviii. Discuss and implement the two possible values of Checkboxes

7.0 REFERENCES/FURTHER READING

[Academic Tutorials \(2008\). 'Academic Tutorials.com'](#).

David, Gowans. (2001). *JavaScript*
<http://www.freewebmasterhelp.com/tutorials/xhtml/>
Marty, Hall. (2009).

JavaScript<http://courses.coreservlets.com/CourseMaterials/pdf/ajax/JavaScript-Core.pdf>

Marty, Hall. (2011). 'HTML: A Crash Course Originals of Slides and Source Code for Examples'.
<http://courses.coreservlets.com/CourseMaterials/ajax.htm.l>

Webmaster. *Tutorial Point, Simply Easy Learning JavaScript Tutorial*.
http://www.tutorialspoint.com/javascript/javascript_tutorial.pdf.

UNIT 4 XML

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 What is XML?
 - 3.2 Differences between XML and HTML
 - 3.3 Elements
 - 3.4 XML Syntax Rules
 - 3.5 Attributes
 - 3.6 Namespaces
 - 3.7 White spaces
 - 3.8 The XML Tree
 - 3.9 Entity
 - 3.9.1 Entity Declaration
 - 3.9.2 The General Entity Declaration
 - 3.9.3 The Parameters Entity Declaration
 - 3.10 Comment
 - 3.11 Viewing XML Files
 - 3.12 A Simple XML Document
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Reading

1.0 INTRODUCTION

XML stands for eXtensible Markup Language. XML was released in the late 90's and has since received a great amount of attention. The XML standard was created by [W3C](#) to provide an easy to use and standardised way to store self-describing data (self-describing data is data that describes both its content and its structure).

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- state the elements of an XML document
- differentiate between the HTML and XML tags
- structure data using XML.

3.0 MAIN CONTENT

3.1 What is XML?

XML is a markup language much like HTML, created to structure, store, and transport information. XML is important for the web as HTML was to the foundation of the web. XML is the most common tool for data transmissions between all sorts of applications.

3.2 Differences between XML and HTML

Table 3.1: Differences between XML and HTML

XML	HTML
XML was designed to transport and store data.	HTML was designed to display data
XML focuses on what data is i.e. its structure	HTML is focused on how data looks.
XML does not have any predefined tags	All HTML tags are predefined.

3.3 Elements

Elements are the building blocks of XML, dividing a document into a hierarchy of regions, each serving a specific purpose. Some elements are containers, holding text or elements; others are empty, marking a place for some special processing.

3.4 XML Syntax Rules

The following general rules guide the creation of valid XML documents:

1. All elements (with the exception of the XML declaration) must have a closing tag.
2. XML tags are case-sensitive.
Example:

```
<imessage>this is correct</imessage>
```

```
<IMESSAGE>this is NOT correct<imessage>
```
3. XML elements must be properly nested.

Example:

```

<note>
  <to>Bakare</to>
  <from>Okunade</from>
  <message>
    <line>My wedding is tomorrow.</line>
  </note>
</message>

```

Fig. 3.1: Nested XML Elements

The above code is improperly nested since `</message>` is supposed to close before `</note>` is closed.

4. XML documents must contain a root element.
5. Attribute values must be quoted.

Example:

```

<note>
  <to>Bina</to>
  <from>Jordan</from>
  <message date="12/12/2013">
    <line>My wedding is tommorrow.</line>
  </message>

```

Fig. 3.2: Quoted Attribute Values

The date attribute of the message element has a value (12/12/2013) which must be enclosed in quotes. Attributes are used to provide additional information about elements. This information is not part of the data.

6. The syntax for XML comments is as follows:
`<!-- This is a comment -->`

3.5 Attributes

In the element start tag you can add more information about the element in the form of attributes. An attribute is a name-value pair. You can use it to add a unique label to an element, place it in a category, add a Boolean flag,

or otherwise associate some short string of data. One reason to use attributes is if you want to distinguish between elements of the same name. You don't always want to create a new element for every situation, so an attribute can add a little more granularity in differentiating between elements. In narrative applications like DocBook or HTML, it's common to see attributes like class and role used for this purpose.

3.6 Namespaces

Namespaces are a mechanism by which element and attribute names can be assigned to groups. They are most often used when combining different vocabularies in the same document.

3.7 Whitespace

You'll notice in my examples, I like to indent elements to clarify the structure of the document. Spaces, tabs, and newlines (collectively called whitespace characters) are often used to make a document more readable to the human eye

3.8 The XML Tree

XML documents form a tree structure that starts at "the root" and branches to "the leaves".

A sample XML document is given below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Eniola</to>
  <from>Ade</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Fig. 3.3: HTML Tree Structure

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the root element of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 child elements of the root (to, from, heading, and body):

```
<to>Eniola</to>
```

```
<from>Ade</from>
```

```
<heading>Reminder</heading>
```

```
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element:

```
</note>
```

You can assume, from this example, that the XML document contains a note to Eniola from Ade. XML documents must contain a root element.

This element is "the parent" of all other elements. The elements in an XML document forms a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have sub elements (child elements):

```
<root>
```

```
<child>
```

```
<subchild>.....</subchild>
```

```
</child>
```

```
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

3.9 Entities

Entities reference data that act as an abbreviation or can be found at an external location. Entities help to reduce the entry of repetitive information and also allow for easier editing by reducing the number of occurrences of data to edit. Entities are placeholders in XML.

You declare an entity in the document prolog or in a DTD, and you can refer to it many times in the document. Different types of entities have

different uses. You can substitute characters that are difficult or impossible to type with character entities. You can pull in content that lives outside of your document with external entities rather than typing it all over again.

3.9.1 Entity Declaration

There are two types of entity declarations: “General” entity declarations and “Parameter” entity declarations.

3.9.1.1 The General Entity Declaration

The types of general entities include:

- INTERNAL (PARSED)
- EXTERNAL (PARSED)
- EXTERNAL (UNPARSED)

INTERNAL (PARSED) GENERAL ENTITY Declaration

Internal parsed entities generally reference text. The correct definition is that they refer to data that an XML processor has to parse.

```
<!ENTITY name "entity_value">
```

Fig. 3.4: Internal (Parsed) General Entity Declaration

where:

- entity value: any character that is not an '&', '%' or ' " ', a parameter entity reference ('%Name;'), an entity reference ('&Name;') or a Unicode character reference.

Example:

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE author [
<!ELEMENT author (#PCDATA)>
<!ENTITY js "Jo Smith">
]>
<author>&js;</author>
```

Fig. 3.5: Example of Internal (Parsed) General Entity Declaration

EXTERNAL (PARSED) GENERAL ENTITY Declaration:

```
<!ENTITY name SYSTEM "URI">
<!ENTITY name PUBLIC "public_ID" "URI">
\]
```

External parsed entities generally reference text. The correct definition is that they refer to data that an XML processor has to parse. External entities are useful for creating a common reference that can be shared between multiple documents. Any changes that are made to external entities are automatically updated in the documents they are referenced.

There are two types of external entities: private, and public. Private external entities are identified by the keyword `system`, and are intended for use by a single author or group of authors. Public external entities are identified by the keyword `PUBLIC` and are intended for broad use.

Fig. 3.6: External (Parsed) General Entity Declaration

where:

- `URI`: In practice, this is a URL where the external parsed entity can be found.
- `public_ID`: This may be used by an XML processor to generate an alternate URI where the external parsed entity can be found. If it cannot be found at this URI, the XML processor must use the normal URI.

Example:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE copyright [
<!ELEMENT copyright (#PCDATA)>
<!ENTITY c SYSTEM "http://www.xmlwriter.net/copyright.xml">
]>
<copyright>&c;</copyright>
```

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE copyright [
<!ELEMENT copyright (#PCDATA)>
<!ENTITY c PUBLIC "-//W3C//TEXT copyright//EN"
    "http://www.w3.org/xmlspec/copyright.xml">
]>
<copyright>&c;</copyright>
```

Fig. 3.7: Example of External (Parsed) General Entity Declaration

EXTERNAL (UNPARSED) GENERAL ENTITY Declaration:

External unparsed entities generally reference non-XML data. The 100% correct definition is that they refer to data that an XML processor does not have to parse.

```
<!ENTITY name SYSTEM "URI" NDATA name>
```

```
<!ENTITY name PUBLIC "public_ID" "URI" NDATA name>
```

Fig. 3.8: External (Unparsed) General Entity Declaration

Example:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE img [
<!ELEMENT img EMPTY>
<!ATTLIST img src ENTITY #REQUIRED>
<!ENTITY logo SYSTEM "http://www.xmlwriter.net/logo.gif" NDATA
gif>
<!NOTATION gif PUBLIC "gif viewer">
]>

```

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE img [
<!ELEMENT img EMPTY>
<!ATTLIST img src ENTITY #REQUIRED>
<!ENTITY logo PUBLIC "-//W3C//GIF logo//EN"
    "http://www.w3.org/logo.gif" NDATA gif>
<!NOTATION gif PUBLIC "gif viewer">
]>
```

```

```

Fig. 3.9: Example of External (Unparsed) General Entity Declaration

Using Entities within Entities:

The following examples show how general entities can be used in the DTD.

CORRECT Example:

```
<?xml version="1.0"?>
<!DOCTYPE author [
<!ELEMENT author (#PCDATA)>
<!ENTITY email "josmith@theworldaccordingtojosmith.com">
<!--the following use of a general entity is legal if it
is used in the XML document-->
<!ENTITY js "Jo Smith &email;">
]>
<author>&js;</author>
```

Fig. 3.10: External (Unparsed) Entities within Entities

INCORRECT Example:

```
<!--the two entity
statements are infinitely recursive-->
<!ENTITY email "user@user.com &js;">
<!ENTITY js "Jo Smith &email;">
```

Fig. 3.11: Incorrect Example

Predefined GENERAL Entities:

Predefined entities are entities already used for markup. The table below lists the predefined entities and how to declare them in a DTD.

Predefined Entities:	How to Declare these Entities in a DTD:
<	<!ENTITY lt "&#60;">
>	<!ENTITY gt "&#62;">
&	<!ENTITY amp "&#38;">

'	<!ENTITY apos "'">
"	<!ENTITY quot """>

Fig. 3.11: Predefined General Entities

3.9.1.2 The Parameter Entity Declaration:

The types of parameter entities include:

- INTERNAL (PARSED)
- EXTERNAL (PARSED)

Rules:

Parameter entity references may not be used within markup in an internal DTD.


INTERNAL (PARSED) PARAMETER ENTITY Declaration:

Internal parameter entity references are used to declare entities existing only in the DTD.

```
<!ENTITY % name "entity_value">
```

Fig. 3.12: Internal (Parsed) Parameter Entity Declaration

where:

entity_value: any character that is not an '&', '%' or ' ' , a parameter entity reference ('%Name;'), an entity reference ('&Name;') or a Unicode  character reference.

Examples:

```
<!--external DTD example-->
<!ENTITY % p "(#PCDATA)">
<!ELEMENT student (id,surname,firstname,dob,(subject)*)>
<!ELEMENT id %p;>
<!ELEMENT surname %p;>
<!ELEMENT firstname %p;>
<!ELEMENT dob %p;>
<!ELEMENT subject %p;>
```

```
<!--external DTD example-->
<!ELEMENT author (#PCDATA)>
<!ENTITY % js "Jo Smith">

<!--note that the general
entity statement below is used
to reference a parameter entity-->
<!ENTITY wb "written by %js;">
```

```
<!--external DTD example-->
<!ENTITY % info "(id,surname,firstname)">
<!ELEMENT lab_group_A %info;>
<!ELEMENT lab_group_B %info;>
<!ELEMENT lab_group_C %info;>
```

Fig. 3.13: Example of Internal (Parsed) Parameter Entity Declaration

Note:

Note the use of external DTD examples above. Parameter entity references may not be used within markup in an internal DTD.

EXTERNAL (PARSED) PARAMETER ENTITY Declaration:

External parameter entity references are used to link external DTDs. There are two types of external entities: private, and public. Private external entities are identified by the keyword **SYSTEM**, and are intended for use by a single author or group of authors.

Public external entities are identified by the keyword **PUBLIC** and are intended for broad use.

```
<!ENTITY % name SYSTEM "URI">
%name;
```

```
<!ENTITY % name PUBLIC "public_ID" "URI">
%name;
```

Fig. 3.14: External (Parsed) Parameter Entity Declaration

where:

- **URI:** In practice, this is a URL where the external parameter entity can be found.
- **public_ID:** This may be used by an XML processor to generate an alternate URI where the external parameter entity can be found. If it cannot be found at this URI, the XML processor must use the normal URI.

Example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE student [
<!ENTITY % student SYSTEM "http://www.university.com/student.dtd">
%student;
]>
```

Fig. 3.15: Example of External (Parsed) Parameter Entity Declaration

3.10 Comments

XML comments use the form you are familiar with from HTML. Comments start with the characters:

```
<!--
```

and end with:

```
-->
```

Anything between these delimiters is ignored by the interpreter. They are notes in the document that are not interpreted by the XML processor.

Comments can be used to put descriptive or historical information into a script, such as:

```
<!-- Script by Eniola and Adekunle, 29th December, 2013. -->
```

Also, comments can be used to disable a section of a document without actually removing it from the entire file by making it inactive having turned it to comment, XML will ignore the execution of such a section turned comment. For instance, you may use the maintainer meta-tag while you are developing a document. When you are ready to deploy it, instead of deleting the tag, you simply comment it out as shown above.

Now you can leave the tag in place; the next time you do some work on the document, you can reactivate the tag by simply removing the comment delimiters.

Caution: Do not use strings of consecutive dash characters (---) inside comments.

3.11 Viewing XML Files

XML files can be viewed with a variety of text editors and all major modern web browsers (mozilla firefox, safari, internet explorer etc.). As will be seen later, looking at a raw XML file isn't anything like looking at an HTML file because as stated earlier, XML files contain no formatting information with along with the data.

Consider the following example:

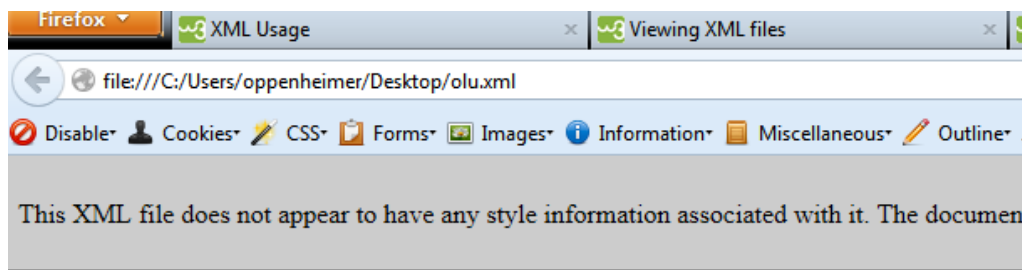
“Olu Bamidele is an employee of company XYZ. He was born on Dec 25th 1990 and is from Lagos state. He works as a production engineer. His salary is 350000 naira monthly”.

In XML, the following data can be marked up as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?><person><firstname>Olu</firstname><lastname>Bamidele</lastname><company>XYZ</company><dob>25/12/1990</dob><stateoforigin>Lagos</stateoforigin><jobtitle>Production Engineer</jobtitle><salary currency="naira">350000</salary></person>
```

Fig. 3.16: Viewing XML Files

As can be seen, this data has no formatting information and would be displayed in a web browser. Some browsers provide some colour coding for the root and child elements as well as for attributes and their values



```
- <person>
  <firstname>Olu</firstname>
  <lastname>Bamidele</lastname>
  <company>XYZ</company>
  <dob>25/12/1990</dob>
  <stateoforigin>Lagos</stateoforigin>
  <jobtitle>Production Engineer</jobtitle>
  <salary currency="naira">350000</salary>
</person>
```

Fig. 3.17: Viewing XML Files**3.12 A Simple XML Document**

```
<article>
<author>Eniola Adekunle</author>
<title>The Web Server Technology </title>
<text>
<abstract>In order to evolve...</abstract>
<section number="1" title="Introduction">
The <index>Web</index> provides the universal...
</section>
</text>
</article>
```

The tags `<article>`, `<author>`, `<abstract>` and `<index>` from the code snippet above are freely definable tags. Tag `<text>` is the start tag and `</text>` is the end tag. Starting from start tag `<text>` and end `</text>` plus its content are called element. And content of starting tag `<text>` and end `</text>` only without the tags itself is called Content of Element or Sub-elements or Text

4.0 CONCLUSION

XML is used in many aspects of web development, often to simplify data storage and sharing. The future might give us word processors, spreadsheet applications and databases that can read each other's data in XML format, without any conversion utilities in between. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

5.0 SUMMARY

In this unit, we looked at various aspects of XML starting with a brief introduction into what it is and how it works. We then examined the differences between XML and HTML. In addition we discussed the structure of XML documents and syntax rules for writing valid XML documents. We ended the unit by taking a look at viewing XML files using web browsers and text editors.

6.0 TUTOR- MARKED ASSIGNMENT

- i. State any five (5) XML syntax rules.
- ii. Markup the following information using XML:

“The Flexmaster9000 is the flagship photocopier from Samsung. It costs \$15000 dollars. The photocopier is always black in color and weighs exactly one tonne. It is capable of photocopying at speeds of up to 100 pages per minute.”

7.0 REFERENCES/FURTHER READING

Benoit, Marchal (2000). *XML By Example*. www.ebay.com/ctg/Applied-XML

W3Schools (2007). XML. *W3Schools*.
<http://www.w3schools.com/xml/default.asp>.

Erik, T. R. (n.d). *Creating Self-Describing Data*. (2nd ed.). Covers W3CXML Schema.

MODULE 3

Unit 1	Perl CGI
Unit 2	PHP
Unit 3	ASPS

UNIT 1 PERL CGI

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	What is a CGI Script?
3.2	Basics of a CGI Program
3.3	The CGI.pm Module
3.4	Perl Variables
3.4.1	Scalars
3.4.2	Arrays
3.5	Simple Form Processing
4.0	Conclusion
5.0	Summary
6.0	Tutor- Marked Assignment
7.0	References/Further Reading

1.0 INTRODUCTION

“CGI” stands for “Common Gateway Interface.” CGI is one method by which a web server can obtain data from (or send data to) databases, documents, and other programs, and presents that data to viewers via the web. More simply, a CGI is a program intended to be run on the web. A CGI program can be written in any programming language, but Perl is one of the most popular.

2.0 OBJECTIVES

At the end of the unit, you should be able to:

- explain what a CGI script is?
- state the building blocks of CGI applications
- analyse form processing applications using Perl CGI.

3.0 MAIN CONTENT

3.1 What is a CGI Script?

Normally, when a web browser looks up a URL the following happens. First your computer contacts the HTTP server with the URL. The HTTP server looks at the filename requested by your computer and then sends that file back. Your computer then displays the file in the appropriate format.

However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your computer to display. This function is called the common gateway interface or CGI. The programs are called CGI scripts.

In summary CGI scripts are programs which can generate and send back anything: sound, pictures, HTML documents, and so on. In this tutorial we concentrate on generating and sending back HTML documents.

3.2 Basics of a CGI Program

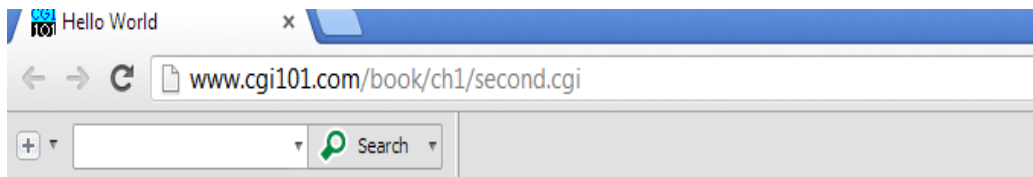
The first task is to write a very simple Perl program which just prints out an HTML document. This should include all the usual `<head> ... </head>` and `<title> ... </title>` commands. However, there are two things to note.

First of all, the first two lines of the print out are special. The first line must be `Content-type: text/html` so that the browser knows what kind of document it is and therefore how to display it. In this case it is a text document of the subclass HTML. You can also send back plain ASCII documents by using `text/plain` instead. Sound samples and images also have to have the content type specified appropriately. The second line must be blank (i.e. it must contain just a line feed). This line is reserved for future development.

The second point is really a tip: Strings in perl programs can span over several lines. This means the main part of your perl program can just be one enormous print statement.

Let's take a look at an example:

```
print "Content-type: text/html\n\n";  
print "<html><head><title>Hello  
world</title></head>\n";  
print "<body>\n";  
print "<h2>Hello, world!</h2>\n";  
print "</body></html>\n";
```



Hello, world!

Figure 1.2:

Print Statement

Output/Result Display

3.3 The CGI.pm Module

Perl offers a powerful feature to programmers: add-on modules. These are collections of pre-written code that you can use to do all kinds of tasks. Some modules are included as part of the Perl distribution; these are called standard library modules and do not have to be installed. If you have Perl, you already have the standard library modules. The CGI.pm module is part of the standard library, and has a number of useful functions and features for writing CGI programs.

Let's see how to use a module in your CGI program. First you have to actually include the module via the use command before any other code:

```
use CGI qw(:standard);
```

Fig. 1.3: CGI.pm Module

Note that even we say use CGI and not use CGI.pm. The .pm is implied in the use statement. The qw(:standard) part of this line indicates that we're importing the "standard" set of functions from CGI.pm.

Now you can call the various module functions by typing the function name followed by any arguments:

```
function name(arguments)
```

If you aren't passing any arguments to the function, you can omit the parentheses.

A function is a piece of code that performs a specific task; it may also be called a subroutine or a method. Functions may accept optional arguments (also called parameters), which are values (strings, numbers, and other variables) passed into the function for it to use. The CGI.pm module has many functions; for now we'll start by using these three:

```
header;  
start_html;  
end_html;
```

The header function prints out the "Content-type" header. With no arguments, the type is assumed to be "text/html". start_html prints out the <html> , <head> , <title> and <body> tags. It also accepts optional arguments. If you call start_html with only a single string argument, it's assumed to be the page title. For example:

```
print start_html("Hello World");  
will print out the following:
```

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

Fig. 1.4: Header Function

Actually start_html prints out a full XML header, complete with XML and DOCTYPE tags.

In other words, it creates a proper HTML header for your page. You can also set the page colours and background image with `start_html`:

```
printstart_html(-title=>"Hello world",
               -bgcolor=>"#cccccc", -text=>"#999999",
```

Fig. 1.5: Print Start_html

Notice that with multiple arguments, you have to specify the name of each argument with `-title=>`, `-bgcolor=>`, etc. This example generates the same HTML as above, only the body tag indicates the page colours and background image:

```
<body        bgcolor="#cccccc"        text="#999999"
background="bgimg.jpg">
```

The `end_html` function prints out the closing HTML tags:

```
</body>
</html>
```

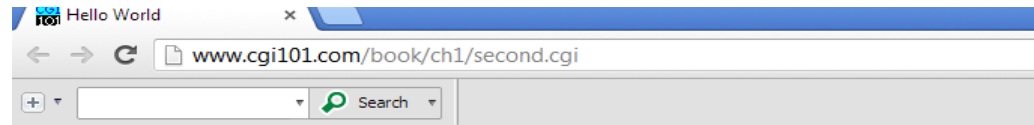
Fig. 1.7: End_html Function

Putting it altogether is the example below:

```
use CGI qw(:standard);
# This is a comment
# Comments are useful for documentation
print header;
printstart_html("Hello world");
print "<h2>Hello, world!</h2>\n";
printend_html; # print the footer
```

Fig. 1.8: Putting it Altogether

The output is:



Hello, world!

Fig. 1.9: Output/Result Display

3.4 Perl Variables

Perl has three types of variables: scalars, arrays, and hashes. However for the purpose of this course, we will be restricting our discussion to scalars and arrays since they are the most commonly used data types in CGI programming.

3.4.1 Scalars

A scalar variable stores a single (scalar) value. Perl scalar names are prefixed with a dollar sign (\$), so for example, \$x, \$username, and \$url are all examples of scalar variable names. Here is how variables are set:

```
$foo = 1;  
$name = "Fred";  
$pi = 3.141592;
```

In this example \$foo,\$name , and \$pi are scalars. You do not have to declare a variable before using it, but it is considered good programming style to do so. There are several different ways to declare variables, but the most common way is with my function:

```
my $foo = 1;  
my ($name) = "Fred";  
my ($pi) = 3.141592;
```

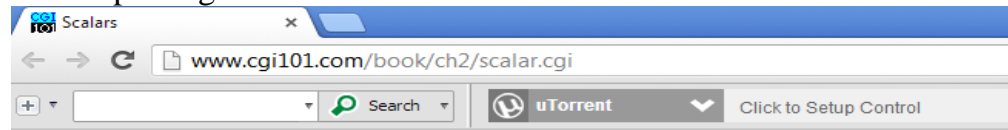
mysimultaneously declares the variables and limits their scope (the area of code that can see these variables) to the enclosing code block.

A scalar can hold data of any type, be it a string, a number, or what not. Let's take a look at an example using scalars:

```
use CGI qw(:standard);  
my $email = "fnord\@cgi101.com";  
my $url = "http://www.cgi101.com";  
print header;  
printstart_html("Scalars");  
<h2>Hello</h2>  
<p>  
My e-mail address is $email, and my web url is <a  
href="$url">$url</a>.  
</p>  
printend_html;
```

Fig. 1.10: Using Scalars

The output is given below:

**Hello**

My e-mail address is `fnord@cgi101.com`, and my web url is <http://www.cgi101.com>.

Fig. 1.11: Output/Result Display Using Scalars

Notice that the @-sign in the e-mail address is escaped with (preceded by) a backslash. This is because the @-sign means something special to Perl – just as the dollar sign indicates a scalar variable, the @-sign indicates an array, so if you want to actually use special characters like @, \$, and % inside a double-quoted string, you have to precede them with a backslash (\).

3.4.2 Arrays

An array stores an ordered list of values. While a scalar variable can only store one value, an array can store many. Perl array names are prefixed with an @-sign. Here is an example:

```
my @colors = ("red","green","blue");
```

Each individual item (or element) of an array may be referred to by its index number. Array indices start with 0, so to access the first element of the array @colors, you use \$colors[0].

To print each element in the @colors array:

```
print "$colors[0]\n"; # prints "red"      print "$colors[1]\n"; # prints
"green"                print "$colors[2]\n"; # prints "blue"
```

Another useful operation that can be performed on arrays is slicing. For instance, to retrieve the first 2 elements of the @colors array and store it in an array called @slice, we could do the following:

```
my @colors = ("red","green","blue");
```

```
my @slice = $colors[0..1];
```

This example sets @slice to ("red","green");

3.5 Simple Form Processing

There are two ways to send data from a web form to a CGI program: GET and POST. These methods determine how the form data is sent to the server.

With the GET method, the input values from the form are sent as part of the URL and saved in the CGI QUERY_STRING environment variable. With the POST method, data is sent as an input stream to the program.

We will take a look at the GET method.

The QUERY_STRING environment variable is set to whatever appears after the question mark (?) in a URL. For instance in the URL <http://www.cgi101.com/book/ch3/env.cgi?test1>, the QUERY_STRING is set to "test1".

You can also process simple forms using the GET method. Below is an example (filename: getform.html):

```
<html>
<head><title>Sample Form</title></head>
<body><form action="env.cgi" method="GET">
First Name: <input type="text" name="fname" size=30><p>
Last Name: <input type="text" name="lname" size=30><p>
<input type="submit">
</form>
</body>
</html>
```

Fig. 1.12: Simple Form Processing

The output is given below with the form filled with 2 data inputs (Joe for first name and Smith for last name):

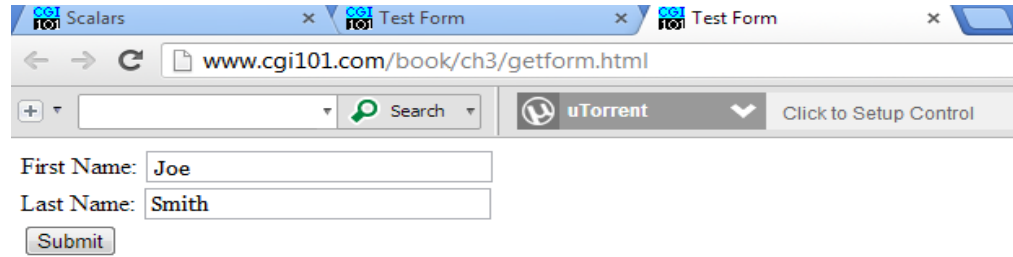


Fig. 1.13: Output Display of Simple Form Processing

This will be passed to the env.cgi program as follows:

```
$ENV{QUERY_STRING} = "fname=Joe&lname=Smith"
```

The two form values are separated by an ampersand (&). You can divide the query string with Perl's split function:

```
my @values = split(/&/,$ENV{QUERY_STRING});
```

“split” lets you break up a string into a list of strings, splitting on a specific character. In this case, we've split on the “&” character. This gives us an array named @values containing two elements: ("fname=Joe", "lname=Smith"). We can further split each string on the “=” character using a foreach loop:

```
foreach my $i (@values) {
  my($fieldname, $data) = split(/=/, $i);
  print "$fieldname = $data<br>\n";
}
```

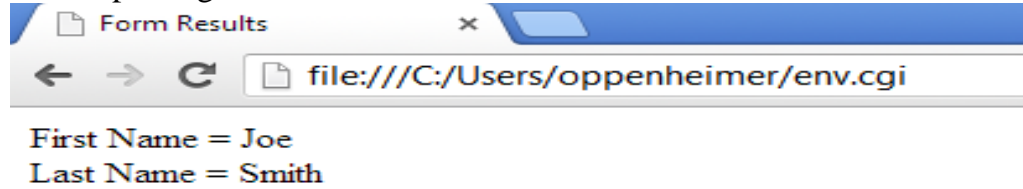
This prints out the field names and the data entered into each field in the form.

The code for env.cgi is given below:

```
use CGI qw(:standard);
print header;
printstart_html("Form Results");
my @values = split(/&/,$ENV{QUERY_STRING});
foreach my $i (@values) {
  my($fieldname, $data) = split(/=/, $i);
  print "$fieldname = $data<br>\n";
}
printend_html;
```

Fig. 1.14: env.cgi Code

The output is given below:

**Fig. 1.15: Output Display of env.cgi Code**

4.0 CONCLUSION

As is the case with many modern programming languages, there is far more to Perl CGI than can be covered in this material. The uses of the language span from web page redirections to MySQL database programming. The keen reader is challenged to explore these additional aspects of the language to fully understand it's potential.

5.0 SUMMARY

In this unit, we look at the basics of CGI programming using Perl. We began with an introduction into what a CGI script is and the basics of CGI scripting using Perl. As part of our discussion, we took a look at scalars and arrays – two of the most commonly used Perl data structures – and how they are used in Perl CGI programming. We rounded up the unit by taking a look at simple form processing using Perl CGI.

6.0 TUTOR-MARKED ASSIGNMENT

Design an address book that site visitors can fill online. After submitting their contact details, a Perl CGI script should display a confirmation message and a summary of all the data the visitor has submitted.

7.0 REFERENCES/FURTHER READING

Hamilton, Jacqueline D. (2004). *CGI Programming 101. Perl for the World Wide Web*, CGI 101.COM

Nik, Silver (2003). *CGI Tutorial*
Online].<http://www.comp.leeds.ac.uk/Perl/Cgi/start>. Html

UNIT 2 PHP

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 What is PHP?
 - 3.2 How PHP Works
 - 3.3 Writing a PHP Page
 - 3.4 PHP Variables
 - 3.5 Simple Form Processing with PHP
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Reading

1.0 INTRODUCTION

PHP was developed in 1994 by Rasmus Lerdorf to track visitors to his online resume, and was released as personal home page tools the following year. This was rewritten and combined with an HTML Form interpreter later that year in PHP/FI version 2. This grew rapidly in popularity and by around the middle of 1997, PHP had ceased to be Rasmus Lerdorf's personal project and had become an important web technology.

PHP script sections are enclosed in `<?php.. ?>` tags and embedded within an HTML page.

These scripts are executed on the server before the page is sent to the browser, so there is no issue of browser-support for PHP pages. PHP is platform-independent, and there are versions for various flavors of windows, unix and linux, and for a number of web servers, including Apache and IIS. The decisive factor is that it's free and open-source.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- analyse how PHP works as a server-side scripting language
- write PHP scripts to process HTML forms.

3.0 MAIN CONTENT

3.1 What is PHP?

PHP, a scripting language designed specifically for use on the web, is your tool for creating dynamic web pages. Rich in features that make web design and programming easier, PHP is in use on over 13 million domains (according to the Netcraft survey at www.php.net/usage.php).

Its popularity continues to grow, meaning that it must be fulfilling its function pretty well.

PHP stands for PHP: HyperText Preprocessor. In its early development by a guy named Rasmus Lerdorf, it was called personal home page tools. When it developed into a full-blown language, the name was changed to be more in line with its expanded functionality.

The PHP language's syntax is similar to the syntax of C but much simpler because it doesn't use some of the more difficult concepts. PHP also doesn't include the low-level programming capabilities of C because PHP is designed to program web sites and doesn't require those capabilities.

PHP is particularly strong in its ability to interact with databases. PHP supports pretty much every database you've ever heard of (and some you haven't). PHP handles connecting to the database and communicating with it. You don't need to know the technical details for connecting to a database or for exchanging messages with it. You tell PHP the name of the database and where it is, and PHP handles the details. It connects to the database, passes your instructions to the database, and returns the database response to you.

3.2 How PHP Works

PHP is an embedded scripting language when used in web pages. This means that PHP code is embedded in HTML code. You use HTML tags to enclose the PHP language that you embed in your HTML file — the same way that you would use other HTML tags.

You create and edit web pages containing PHP the same way that you create and edit regular HTML pages.

The PHP software works in conjunction with the web server. The web server is the software that delivers web pages to the world. When you type

a URL into your web browser, you're sending a message to the web server at that URL, asking it to send you an HTML file. The web server responds by sending the requested file. Your browser reads the HTML file and displays the web page. You also request the web server to send you a file when you click a link in a web page. In addition, the Web server processes a file when you click a Web page button that submits a form.

When PHP is installed, the web server is configured to expect certain file extensions to contain PHP language statements. Often the extension is .php or .phtml, but any extension can be used. When the web server gets a request for a file with the designated extension, it sends the HTML statements as-is, but PHP statements are processed by the PHP software before they're sent to the requester.

When PHP language statements are processed, only the output is sent by the web server to the web browser. The PHP language statements are not included in the output sent to the browser, so the PHP code is secure and transparent to the user. For instance, in this simple PHP statement:

```
<?php echo "<p>Hello world"; ?>
```

Fig. 2.1: Echo

<?php is the PHP opening tag, and ?> is the closing tag. echo is a PHP instruction that tells PHP to output the upcoming text. The PHP software processes the PHP statement and outputs this:

```
<p>Hello World
```

This is a regular HTML statement. This HTML statement is delivered to the browser. The browser interprets the statement as HTML code and displays a web page with one paragraph — hello world. The PHP statement is not delivered to the browser, so the user never sees any PHP statements.

PHP and the web server must work closely together. PHP is developed as a project of the Apache software foundation — consequently, it works best with Apache. PHP also works with Microsoft IIS/ PWS, iPlanet (formerly Netscape Enterprise Server), and others.

3.3 Writing A PHP Page

Before writing any PHP code, a web server and the PHP programming language must be installed on the user machine. Installation is platform-

dependent so we encourage readers to explore some of the specifics of installation based on the type of operating system they use. For windows, linux and mac OS users, a popular package, XAMPP, installs PHP, MySQL and Apache together with the click of a button and requires minimal configuration out of the box. XAMPP is available for download at <http://www.apachefriends.org/en/xampp.html>. The page comes with instructions on how to perform the installation depending on the user's operating system.

Once installed, a web documents folder is created. This is typically named htdocs or www. It is within this folder that all PHP documents must be saved to be accessed within the web browser. PHP files can be written using any text-editor and typically would be saved with a .php extension. To run a PHP script, simply type in the URL that point to the script. For instance, if a file called test.php is saved in the htdocs or www folder of a machine, it can usually be accessed by typing: <http://localhost/test.php> in any installed web browser.

The structure of a PHP file is simple. It begins with a `<?php` tag and ends with a `?>` tag. In between these tags, PHP code is embedded.

Let's take a look at the example below:

```
<?php
//filename: test.php
echo "Generated by php";
?>
```

// is used to mark the start of a comments line in PHP. Each line of a PHP script (with the exception of the opening and closing tags) ends with a semicolon. The echo statement is used for output in PHP similar.

The result of the code above is displayed in a web page below. Take note of the URL.

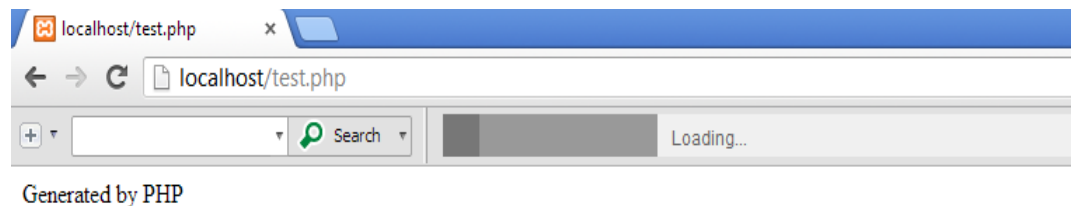


Fig. 2.3: Echo Result/Output Display

3.4 PHP Variables

As with algebra, PHP variables can be used to hold values ($x=5$) or expressions ($z=x+y$).

Variable can have short names (like x and y) or more descriptive names (age, car name, total volume).

The rules for PHP variables are as follows:

- A variable starts with the \$ sign, followed by the name of the variable
- A variable name must begin with a letter or the underscore character
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- A variable name should not contain spaces
- Variable names are case sensitive (\$y and \$Y are two different variables)
- Variables do not need to be declared before they are used. PHP automatically converts the variable to the correct data type, depending on its value.

Let us take a look at some code below:

```
<?php
//filename: vars.php
$x=2;
$y=4;
$sum=$x+$y;
echo $sum."<br />";
$product=$x*$y;
echo $product."<br />";
$quotient=$y/$x;
echo $quotient."<br />";
$string1="This is";
$string2= "a string";
echo $string1." ".$string2;
?>
```

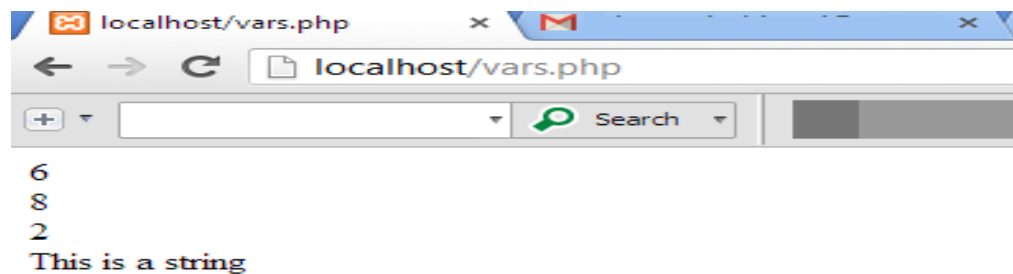
Fig. 2.4: Variable

The code assigns the values of 2 and 4 to variables x and y and proceeds to print their sum, product and quotient. The dot (.) operator is used to concatenate the outputs printed by the echo statement. The `
` added at the end of each echo statement makes sure the next output starts on a new line.

The second part of the code concatenates two string variables together and prints out their concatenation.

As can be seen, PHP correctly determines the type of a variable after a value has been assigned to it. This increases its flexibility and is one of the reasons it is popular among beginners.

The output is given below:

**Fig. 2.5: Variable Content Display****3.5 Simple Form Processing With PHP**

One of the most common uses of PHP on the web is for the processing of HTML forms. PHP can be used to access submitted form data, process the data, store the data in a database and retrieve the data. For the purpose of these lecture notes, we will focus on accessing and processing form data using PHP.

Variables can also be used to store information that is entered by the user via an HTML form. We simply assign a name to the form element where the user will enter data, and the data entered will be available in PHP script as a variable with the same name as the form element (preceded, of course, by a dollar sign).

While this is convenient, it is sometimes necessary to process the user's data in a separate page. To show how this works, we'll develop the `userform.php` example to be spread over two pages: the user will enter a name on the first page and be greeted with a personalised message on the second. The first page, `userform.html`, simply contains the HTML form where the user can input a name. The `ACTION` attribute of the form is set to `processform.php`, which will be the name of our PHP page:

```
<HTML>
<!-- userform.html -->
<FORM ACTION="processform.php" METHOD=POST>
    Please type your name here:<BR>
<INPUT TYPE=TEXT NAME="username"><BR><BR>
<INPUT TYPE=SUBMIT VALUE="Submit data">
</FORM>
```

Fig. 2.6: Form Action

When the submit button is pressed, a request for the `processform.php` page is passed from the browser to the server. Since we have specified the `post` method here, the data will be included in the HTTP header rather than appended to the URL. In PHP, form data is stored in arrays depending on the method used. In this instance, the data is stored in the `$_POST[]` array. To access any element of the array, we use its name.

Thus for a textbox named `username`, its contents will be available using the `$_POST['username']` variable.

```
<?php
//filename: processform.php
echo "welcome, " . $_POST['username'] . "!";
?>
```

Fig. 2.7: // File Action

This page simply writes a personalised "Welcome" message to the browser. Sending data to every new page in this way (e.g. by programmatically appending it to the URL or by including it in a hidden form element), we have a crude means of persisting data between pages.

4.0 CONCLUSION

As is the case with many modern programming languages, there is far more to PHP than can be covered in this material. The uses of the language span from web page redirections to MySQL database programming. The keen reader is challenged to explore these additional aspects of the language to fully understand its potential.

5.0 SUMMARY

In this unit, we took a look at the basics of PHP focusing on how PHP facilitates web interaction. We explored the use of PHP variables and concluded by taking a look at form processing using PHP.

6.0 TUTOR-MARKED ASSIGNMENT

Design an address book that site visitors can fill online. After submitting their contact details, a PHP script should display a confirmation message and a summary of all the data the visitor has submitted.

7.0 REFERENCES/FURTHER READING

Castagnetto, J. *et al.* (1994). *Professional PHP Programming*. Wrox Press, USA.

Valade, J. (2004). *PHP/MYSQL for Dummies*. (2nded.). Wiley Publishing Inc, USA.

UNIT 3 ASP

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 How ASP Works
 - 3.2 Writing an ASP Page
 - 3.3 Passing Variables in a URL
 - 3.4 Simple Form Processing with ASP
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor -Marked Assignment
- 7.0 References/Further Reading

1.0 INTRODUCTION

ASP is an acronym for Active Server Pages. ASP is a technology developed by Microsoft primarily for web scripting. To describe what an ASP page is; you could say that it is a file with the extension .asp that contains a combination of HTML tags and scripts that run on a web server.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- state how ASP works as a Server Side Programming Language
- Write ASP Programming Language that retrieve input from HTML forms
- pass variables in a Url.

3.0 MAIN CONTENT

3.1 How Does ASP Work?

The best way to explain how ASP works is by comparing it with standard HTML.

Imagine you type the address of an HTML document (e.g. <http://www.mysite.com/page.htm>) in the address line of the browser.

This way you request an HTML page. The server simply sends an HTML file to the client. But if you instead type `http://www.mysite.com/page.asp` - and thus request an ASP page - the server is put to work.

The server first reads the ASP file carefully to see if there are any tasks that need to be executed. Only when the server has done what it is supposed to do, the result is then sent to the client. It is important to understand that the client only sees the result of the server's work - not the actual instructions.

This means that if you click "view source" on an ASP page, you do not see the ASP codes - only basic HTML tags. Therefore, you cannot see how an ASP page is made by using "view source".

3.2 Writing an ASP Page

ASP is a server-side technology. Therefore, you need to have a server to run ASP. One way to go about this is to enable the internet information server (IIS) on windows. This can be achieved using the following steps:

1. Go to the control panel
2. Click on Programs and then click turn windows features on or off.
3. In the windows features dialog box, click internet information services and then click OK.

And that's it. IIS now runs on your machine. Your webroot - where all your .asp pages should be located -- is at `C:\inetpub\wwwroot`. To test if your IIS/ASP installation works you have to type in the name of the computer in a web browser. Your browser should look like the image below confirming that the server is running.



Fig. 3.1: Confirming of Server Running

ASP pages typically start with `<%` and end with `%>`. The `<%` is called an opening tag, and the `%>` is called a closing tag. In between these tags are the server-side scripts. You can insert server-side scripts anywhere in your web page--even inside HTML tags.

Let us take a look at a simple ASP page below:

```
<html>
<head>
<title>My first ASP page</title>
</head>
<body>

    <%
    'My first ASP page

    Response.Write "<h1>Hello World!</h1>"

    %>

</body>
</html>
```

In a web browser, the page is displayed as follows:

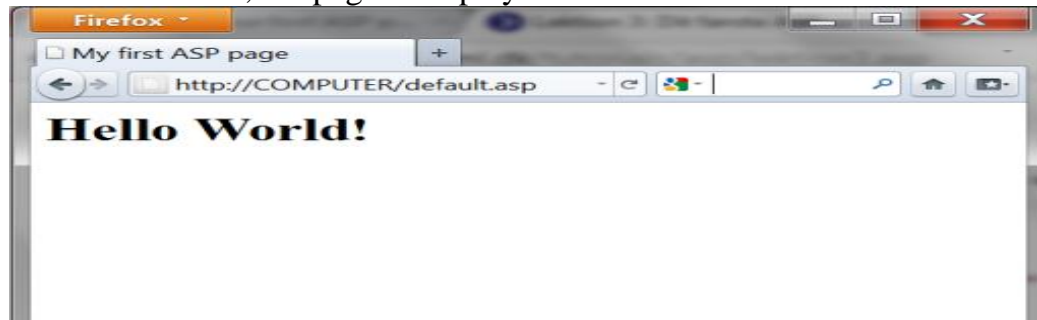


Fig. 3.3: Simple ASP Page Display

As is evident from the browser's output is the Response. Write ASP's print statement. The apostrophe (') is used to mark comments and thus the line, 'My first ASP page, is a comment and consequently is not executed by the ASP server.

3.3 Passing Variables in a Url

When you work with ASP, you often need to pass variables from one page to another.

Maybe you have wondered why some URLs look something like this:

```
http://html.net/page.asp?fname=Joe
```

Fig. 3.4: Passing Variables in a Url

The answer is that the characters after the question mark are a **HTTP query string**. A HTTP query string can contain both variables and their values. In the example above, the HTTP query string contains a variable named `fname`, with the value `Joe`.

In ASP, to fetch the value of a variable passed in a URL, we make use of the `Request.QueryString` Method.

Thus supposing we wanted to print the value of `fname` (in this case, `Joe`) from the URL above; the code would be as follows:

```
<html>
<body>
<%
Response.WriteRequest.QueryString("fname")
%>
</body>
</html>
```

Fig. 3.5: Printing Variable Value from Url

When multiple variables are passed in the same URL, they are separated by `&`. Thus:

```
http://html.net/page.asp?fname=Joe&lname=Fraser&age=35
```

Fig. 3.6: Passing Multiple Variables in the Same URL

The code above has 3 variables (`fname`, `lname` and `age`) with values of `Joe`, `Fraser` and `35` respectively. In ASP, the values of these variables would be accessed separately as follows:

```
<html>
<body>
<%
Response.WriteRequest.QueryString("fname")
Response.WriteRequest.QueryString("lname")
Response.WriteRequest.QueryString("age")
%>
</body>
</html>
```

Fig. 3.7: Accessing Values of Variables Separately**3.4 Simple Form Processing with ASP**

One of the most useful functions of ASP on the web is for processing of form input by users. In order to do this, ASP makes use of Request.QueryString and Request.Form commands to retrieve user input from forms.

As we saw in the previous section, Request.QueryString is used to retrieve values stored in variables passed as part of a URL. This method of passing form variables is employed when GET method of form processing is used. To illustrate this point, let us look at an example below:

```
<html><head><title>A GET FORM</title></head><body><form method="get"
action="simpleform.asp">
First Name: <input type="text" name="fname"><br>
Last Name: <input type="text" name="lname"><br><br>
<input type="submit" value="Submit">
</form></body></html>
```

This form is processed by simpleform.asp displayed below:

```
<body>
Welcome
<%
Response.write(Request.QueryString("fname"))
Response.write(" " &Request.QueryString("lname"))
%>
</body>
```

Information sent from the form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

If a user typed "Bill" and "Gates" in the HTML form above, the URL sent to the server would look like this:

```
http://mycomputer/simpleform.asp?fname=Bill&lname=Gates
```

Fig. 3.10: URL Sent to Server

The browser will display the following in the body of the document:

```
welcome Bill Gates
```

Fig. 3.11: Browser will Display

The Request. Form command is used to collect values in a form with the POST method information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send. To make use of the post method, the HTML form above would be identically written except that the line `method="get"` would be replaced with `method="post"`.

If a user typed "Bill" and "Gates" in the HTML form above, the URL sent to the server would look like this:

```
http://mycomputer/simpleform.asp
```

Fig. 3.12: UrlSent to Server from HTML

As is evident, the form entries are no longer passed along with the URL but they are still present and can be retrieved using the request form method. We modify simple form.asp to make retrieve the data sent using the POST method as follows:

```
<html>
<body>
welcome
<%
Response.write(Request.Form("fname"))
Response.write(" " &Request.Form("lname"))
%>
</body>
</html>
```

Fig. 3.13: Request Form Method

The browser will display the following in the body of the document:

Welcome Bill Gates

Fig. 3.14: Browser Display Request Form**4.0 CONCLUSION**

As is the case with many modern programming languages, there is far more to ASP than can be covered in this material. The uses of the language span from web page redirections to database programming.

The keen reader is challenged to explore these additional aspects of the language to fully understand its potential.

5.0 SUMMARY

In this unit, we took a look at the basics of ASP focusing on how ASP facilitates web interaction. We also explored the use of ASP in retrieving input from HTML forms.

6.0 TUTOR- MARKED ASSIGNMENT

Design an address book that site visitors can fill online. After submitting their contact details, an ASP script should display a confirmation message and a summary of all the data the visitor has submitted.

7.0 REFERENCES/FURTHER READING

ASPTutorial.info, Active Server Pages Tutorials for Beginners (2002).
[Online] www.asptutorial.info

HTML.net, ASP Tutorial, (2005). [Online]
<http://html.net/tutorials/asp/> W3Schools,
<http://www.w3schools.com/asp/default.asp>.

MODULE 4 DATABASE OPERATIONS

Unit 1	Implementation of Database Internals
Unit 2	An Overview of Database Operations
Unit 3	Integrated Web Application

UNIT 1 IMPLEMENTATION OF DATABASE INTERNALS

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	Relational Systems: The Life of a Query
3.2	Relational Query Processor
3.2.1	Query Parsing and Authorisation
3.2.2	Query Rewriter
3.2.3	Query Optimiser
3.2.4	Query Executor
3.2.5	Access Methods
4.0	Conclusion
5.0	Summary
6.0	Tutor- Marked Assignment
7.0	References/Further Reading

1.0 INTRODUCTION

Database Management Systems (DBMSs) are complex, mission-critical software systems. Today's DBMSs embody decades of academic and industrial research and intense corporate software development.

In this unit, we attempt to capture the main architectural aspects of modern database systems, with a discussion of advanced topics. Where applicable, we use commercial and open-source systems as examples of the various architectural forms discussed.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- identify the components of a generic database system
- explain briefly the mechanics of query execution in a database.

3.0 MAIN CONTENT

3.1 Relational Systems: The Life of a Query

The most mature and widely used database systems in production today are relational database management systems (RDBMSs). These systems can be found at the core of much of the world's application infrastructure including e-commerce, medical records, billing, human resources, payroll, customer relationship management and supply chain management, to name a few. The advent of web-based commerce and community-oriented sites has only increased the volume and breadth of their use. Relational systems serve as the repositories of record behind nearly all online transactions and most online content management systems (blogs, wikis, social networks, and the like). In addition to being important software infrastructure, relational database systems serve as a well-understood point of reference for new extensions and revolutions in database systems that may arise in the future. As a result, we focus on relational database systems throughout this unit.

At heart, a typical RDBMS has five main components, as illustrated in Figure 1.1. As an introduction to each of these components and the way they fit together, we step through the life of a query in a database system.

Consider a simple but typical database interaction at an airport, in which a gate agent clicks on a form to request the passenger list for a flight. This button click results in a single-query transaction that works roughly as follows:

1. The personal computer at the airport gate (the "client") calls an API that in turn communicates over a network to establish a connection with the Client Communications Manager of a DBMS (top of Figure 1.1). In some cases, this connection is established between the client and the database server directly, e.g., via the ODBC or JDBC connectivity protocol.

This arrangement is termed a “two-tier” or “client-server” system. In other cases, the client may communicate with a “middle-tier server” (a web server, transaction processing monitor, or the like), which in turn uses a protocol to proxy the communication between the client and the DBMS.

This is usually called a “three-tier” system. In many web-based scenarios there is yet another “application server” tier between the web server and the DBMS, resulting in four tiers. Given these various options, a typical DBMS needs to be compatible with many different connectivity protocols used by various client drivers and middleware systems.

At base, however, the responsibility of the DBMS’ client communications manager in all these protocols is roughly the same: to establish and remember the connection state for the caller (be it a client or a middleware server), to respond to SQL commands from the caller, and to return both data and control messages (result codes, errors, etc.) as appropriate. In our simple example, the communications manager would establish the security credentials of the client, set up state to remember the details of the new connection and the current SQL command across calls, and forward the client’s first request deeper into the DBMS to be processed.

2. Upon receiving the client’s first SQL command, the DBMS must assign a “thread of computation” to the command. It must also make sure that the thread’s data and control out- puts are connected via the communications manager to the client. These tasks are the job of the DBMS Process Manager (left side of Figure 1.1). The most important decision that the DBMS needs to make at this stage in the query regards admission control: whether the system should begin processing the query immediately, or defer execution until a time when enough system resources are available to devote to this query.

3. Once admitted and allocated as a thread of control, the gate agent’s query can begin to execute. It does so by invoking the code in the Relational Query Processor (center, Figure 1.1). This set of modules checks that the user is authorised to run the query, and compiles the user’s SQL query text into an internal query plan. Once compiled, the resulting query plan is handled via the plan executor. The plan executor consists of a suite of “operators” (relational algorithm implementations) for executing any query. Typical operators implement relational query processing tasks including joins, selection, projection, aggregation, sorting and so on, as well as calls to request data records from lower layers of the system. In our example query, a small subset of these operators — as assembled by the query optimisation process — is invoked to satisfy the gate agent’s query.

4. At the base of the gate agent's query plan, one or more operators exist to request data from the database. These operators make calls to fetch data from the DBMS' Transactional Storage Manager (Figure 1.1, bottom), which manages all data access (read) and manipulation (create, update, delete) calls. The storage system includes algorithms and data structures for organizing and accessing data on disk ("access methods"), including basic structures like tables and indexes. It also includes a buffer management module that decides when and what data to transfer between disk and memory buffers. Returning to our example, in the course of accessing data in the access methods, the gate agent's query must invoke the transaction management code to ensure the well-known "ACID" properties of transactions. Before accessing data, locks are acquired from a lock manager to ensure correct execution in the face of other concurrent queries. If the gate agent's query involved updates to the database, it would interact with the log manager to ensure that the transaction was durable if committed, and fully undone if aborted.

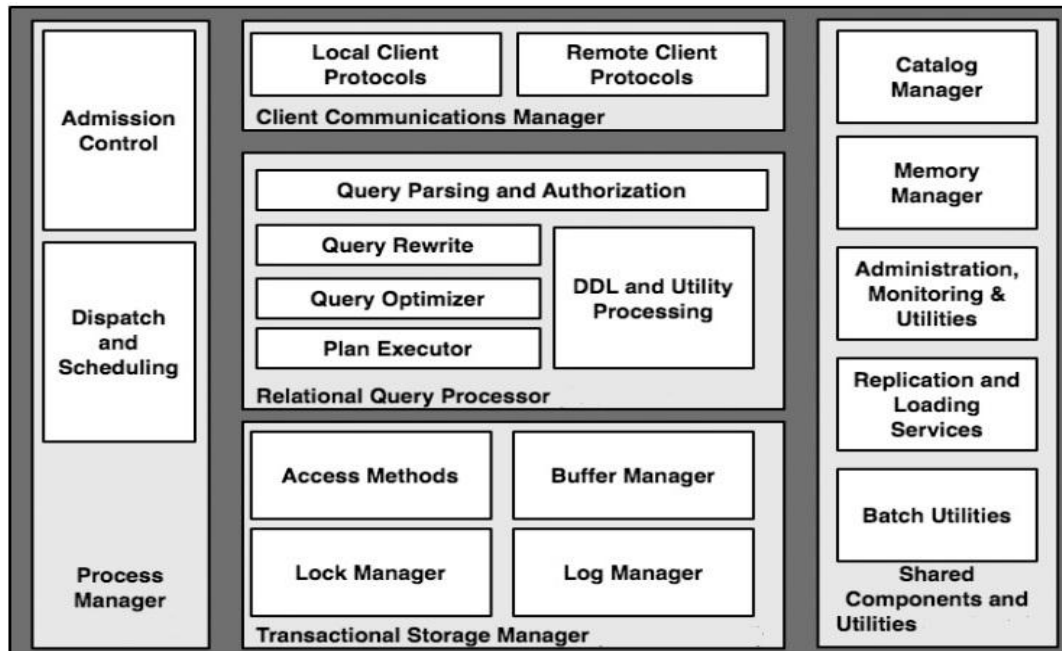
5. At this point in the example query's life, it has begun to access data records, and is ready to use them to compute results for the client. This is done by "unwinding the stack" of activities we described up to this point. The access methods return control to the query executor's operators, which orchestrate the computation of result tuples from database data; as result tuples are generated, they are placed in a buffer for the client communications manager, which ships the results back to the caller. For large result sets, the client typically will make additional calls to fetch more data incrementally from the query, resulting in multiple iterations through the communications manager, query executor, and storage manager. In our simple example, at the end of the query the transaction is completed and the connection closed; this results in the transaction manager cleaning up state for the transaction, the process manager freeing any control structures for the query, and the communications manager cleaning up communication state for the connection.

Our discussion of this example query touches on many of the key components in an RDBMS, but not all of them. The right-hand side of Figure 1.1 depicts a number of shared components and utilities that are vital to the operation of a full-function DBMS.

The catalogue and memory managers are invoked as utilities during any transaction, including our example query. The catalogue is used by the query processor during authentication, parsing, and query optimisation. The memory manager is used throughout the DBMS whenever memory needs

to be dynamically allocated or reallocated. The remaining modules listed in the rightmost box of Figure 1.1 are utilities that run independently of any particular query, keeping the database as a whole well-tuned and reliable.

Fig. 1.1: Main Components of a DBMS



3.2 Relational Query Processor

A relational query processor takes a declarative SQL statement, validates it, optimizes it into a procedural dataflow execution plan, and (subject to admission control) executes that dataflow program on behalf of a client program. The client program then fetches (“pulls”) the result tuples, typically one at a time or in small batches. The major components of a relational query processor are shown in Figure 1.1.

3.2.1 Query Parsing and Authorisation

Given an SQL statement, the main tasks for the SQL Parser are to:

1. Check that the query is correctly specified
2. Resolve names and references
3. Convert the query into the internal format used by the optimiser
4. Verify that the user is authorised to execute the query.

Some DBMSs defer some or all security checking to execution time but, even in these systems, the parser is still responsible for gathering the data

needed for the execution-time security check. Given an SQL query, the parser first considers each of the table references in the FROM clause. It canonicalises table names into a fully qualified name of the form `server.database.schema.table`. After canonicalising the table names, the query processor then invokes the catalog manager to check that the table is registered in the system catalog. If the query parses successfully, the next phase is authorisation checking to ensure that the user has appropriate permissions (SELECT/DELETE/INSERT/UPDATE) on the tables, user defined functions, or other objects referenced in the query. If a query parses and passes validation, then the internal format of the query is passed on to the query rewrite module for further processing.

3.2.2 Query Rewrite

The query rewrite module, or rewriter, is responsible for simplifying and normalizing the query without changing its semantics. It can rely only on the query and on metadata in the catalog, and cannot access data in the tables. The rewriter in many commercial systems is a logical component whose actual implementation is in either the later phases of query parsing or the early phases of query optimisation. In DB2, for example, the writer is a stand-alone component, whereas in SQL Server the query rewriting is done as an early phase of the Query Optimiser.

3.2.3 Query Optimiser

The query optimiser's job is to transform an internal query representation into an efficient query plan for executing the query (Figure 4.1). A query plan can be thought of as a dataflow diagram that pipes table data through a graph of query operators. In many systems, queries are first broken into SELECT-FROM-WHERE query blocks. The optimisation of each individual query block is then done using techniques similar to those described in the famous paper by Selinger et al. on the System R optimiser. On completion, a few operators are typically added to the top of each query block as post-processing to compute GROUP BY, ORDER BY, HAVING and DISTINCT clauses if they exist. The various blocks are then stitched together in a straightforward fashion.

To enable cross-platform portability, every major DBMS now compiles queries into some kind of interpretable data structure. The only difference between them is the intermediate form's level of abstraction. The query plan in some systems is a very lightweight object, not unlike a relational algebraic expression, that is annotated with the names of access methods, join algorithms, and so on. Other systems use a lower-level language of

“op-codes,” closer in spirit to Java byte codes than to relational algebraic expressions.

3.2.4 Query Executor

The query executor operates on a fully-specified query plan. This is typically a directed dataflow graph that connects operators that encapsulates base-table access and various query execution algorithms. In some systems, this dataflow graph is already compiled into low-level op-codes by the optimiser. In this case, the query executor is basically a runtime interpreter.

In other systems, the query executor receives a representation of the dataflow graph and recursively invokes procedures for the operators based on the graph layout.

3.2.5 Access Methods

Access methods are the routines that manage access to the various disk-based data structures that the system supports. These typically included unordered files (“heaps”), and various kinds of indexes. All major commercial systems implement heaps and B+-tree indexes. Both Oracle and PostgreSQL support hash indexes for equality lookups.

All DBMSs need some way to “point” to rows in a base table, so that index entries can reference the rows appropriately. In many DBMSs, this is implemented by using direct row IDs (RIDs) that are the physical disk addresses of the rows in the base tables. This has the advantage of being fast, but has the downside of making base table row movement very expensive since all secondary indexes that point to this row require updating. Both finding and updating these rows can be costly. Rows need to move when an update changes the row size and space is unavailable on the current page for the freshly updated row. And many rows need to move when a B+-tree is split. DB2 uses a forwarding pointer to avoid the first problem. This requires a second I/O to find a moved page, but avoids having to update the secondary index. DB2 avoids the second problem by simply not supporting B+-trees as primary storage for base table tuples.

Microsoft SQL Server and Oracle support B+- trees as primary storage and must be able to deal with row movement efficiently. The approach taken is to avoid using a physical row address in the secondary indexes and instead use the row primary key (with some additional system provided bits to force uniqueness if the table does not have a unique key) rather than the physical RID.

4.0 CONCLUSION

The details of the internals of a database are more technical than can be covered within the scope of this unit. Furthermore, a thorough analysis of the different implementation internals will span tomes of technical documentation which are more suited for technical papers. The overview presented in this unit serves as a sufficient introduction for further study in more advanced courses

5.0 SUMMARY

In this unit we took a look at the technical architecture of a generic database system looking at the broad functions of the specific parts. We then drilled down on the mechanics of relational query execution looking at the various processes involved in query execution.

6.0 TUTOR-MARKED ASSIGNMENT

Identify any 5 components of a database system, stating briefly 2 functions of each component.

7.0 REFERENCES/FURTHER READING

Ailamaki, A. *et al.* (2003). *Exposing Undergraduate Students to Database System Internals*, SIGMOD Record, and Vol. 32, No. 3.

Hellerstein, J. *et al.* (2007). *Architecture of a Database System, Foundations and Trends in Databases*.

UNIT 2 AN OVERVIEW OF DATABASE OPERATIONS

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 What is SQL?
 - 3.2 SQL Commands
 - 3.2.1 Data Definition Language
 - 3.2.2 Data Manipulation Language
 - 3.2.3 Data Control Language
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor- Marked Assignment
- 7.0 References/Further Reading

1.0 INTRODUCTION

SQL, an acronym for **Structured Query Language**, is the language of databases. It includes database commands for creation, deletion, fetching and modifying database records and so on. SQL is an ANSI (American National Standards Institute) standard but there are many different versions of SQL language.

2.0 OBJECTIVES

At the end of the unit, you should be able to:

- explain briefly what sql is
- categorise sql commands as either data manipulation, data definition or data control languages
- describe the basic syntax of the most commonly used SQL commands.

3.0 MAIN CONTENT

3.1 What Is SQL?

SQL (Structured Query Language) is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS).

Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language and a data manipulation language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control.

SQL was one of the first commercial languages for Edgar F. Codd's relational model, as described in his influential 1970 paper: A Relational Model of Data for Large Shared Data Banks. Despite not entirely adhering to the relational model as described by Codd, it has become the most widely used database language.

SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. This version, initially called SEQUEL (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM's database management system, System R, developed during the 1970s. The acronym SEQUEL was later changed to SQL because "SEQUEL" was an existing trademark.

In the late 1970s, Relational Software, Inc. (now Oracle Corporation) saw the potential of the concepts described by Codd, Chamberlin, and Boyce and developed their own SQL-based RDBMS. In June 1979, Relational Software, Inc. introduced the first commercially available implementation of SQL, Oracle V2 (Version2) for VAX computers. Since then other vendors including Microsoft and MySQL AB have developed independent implementations of SQL usually with added features for differentiation from competitors.

3.2 SQL Commands

SQL commands are divided into 3 categories based on their nature:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)

3.2.1 Data Definition Language (DDL)

This subset of SQL commands are used to define data structures and in particular, database schemas. Below is a table that summarises the DDL commands, their descriptions and their basic syntax.

Table 2.1: Data Definition Language (DDL)

COMMAND	DESCRIPTION	SYNTAX
CREATE	Creates an object (a table, for example) in the database.	CREATE TABLE <i>tablename</i> (<i>field1</i> <i>datatype</i> (<i>length</i>), <i>field2</i> <i>datatype</i> (<i>length</i>),... <i>fieldn</i> <i>datatype</i> (<i>length</i>) PRIMARY KEY (<i>fieldk</i>)
ALTER	Modifies an existing database object, such as a table.	ALTER <i>objecttypeobjectna</i> <i>meparameters</i> .
DROP	Deletes an entire table, a view of a table or other object in the database.	DROP <i>objecttypeobjectna</i> <i>me</i> .

Let us take a look at these SQL commands in action. The task is to create a table with 5 columns to store the first name, surname, employee number, job title and gender of employees at a company.

First we create a database. After creating the database, we create tables within the database.

To create the database:

```
CREATE DATABASEpractice;
```

Fig. 2.1: To Create the Database

To create the table, we use the following command:

```
CREATE TABLE employee (
employee_numberINTfirst_nameVARCHAR(50),
surname VARCHAR(50),job_titleVARCHAR(70),
gender VARCHAR(6) PRIMARY KEY
(employee_number)
```

Fig. 2.2: To Create the Table

The result is given below:

Table name: employee

Table 2.2: The Result of the Table Created above

employee_number	first_name	Surname	job_title	Gender

Our create command creates an empty table as illustrated above. The data type for employee number is specified as INT (short for INTEGER) indicating that it stores integer values. For columns that contain strings, we use the varchar data type. We specify the lengths based on the anticipated largest number of characters that a table record can contain for each column. We set the primary key to the employee number field because we expect that employee numbers uniquely identify employees of the company and thus no two employees can have the same employee number.

Let's take a look at another example. Supposing we want to remove the job title column from our employee table and add a column for date of birth, we would proceed as follows:

```
ALTERTABLE employee DROP COLUMN job_title;
ALTER TABLE employee ADD date_of_birth DATE;
```

Fig. 2.3: Commands to Alter the Table Created above

As can be seen from the queries above, first we remove (or DROP) the job_title column and then add a column, date_of_birth with data type DATE. The result is given below:

Table 2.3: Result/Output of the above Altered Table

employee_number	first_name	surname	Gender	date_of_birth

To delete the table and the database create above, we execute the following commands:

```
DROP TABLE employee;
DROP DATABASE practice;
```

3.2.2 Data Manipulation Language (DML)

This subset of SQL commands is used for reading (selecting), inserting, deleting and updating data in a database. Below is a table that summarises the DML commands, their descriptions and their basic syntax:

Table 2.4: Data Manipulation Language (DML)

COMMAND	DESCRIPTION	SYNTAX
SELECT	Retrieves certain records from one or more tables	SELECT <i>field1, field2, ..., fieldn</i> FROM <i>table1, table2, ... tablen</i> [WHERE <i>condition</i>][ORDER BY <i>fieldk</i>]
INSERT	This statement adds one or more records to any table in a relational database.	INSERT INTO table(<i>column1, column2, ... columnn</i>) VALUES (<i>value1, value2, valuen</i>)
UPDATE	This statement changes the data of one or more records in a table.	UPDATE <i>table_name</i> SET <i>column1 = value,</i> <i>column2=value2, ...,</i> <i>column=valuen</i> [WHERE <i>condition</i>]
DELETE	This statement removes one or more records from a table.	DELETE FROM <i>table_name</i> [WHERE <i>condition</i>]

Let us take a look at these SQL commands in action. With the employee table from the previous section, let us select all employees whose surnames are “Mundi”.

Table 2.5: Employees Table

employee_number	first_name	Surname	gender	date_of_birth
0001	Yashim	Mundi	M	12-12-1980
0002	Jacob	Adewale	M	12-12-1988
0003	Akim	Ellah	M	14-05-1970
0004	Takon	Mundi	F	16-09-1985
0005	Abdul	Mohammed	M	18-02-1972

The SELECT query to select all employees whose surnames are “Mundi” will be as follows:

```
SELECT * FROM employee WHERE surname="Mundi";
```

Fig. 2.5: Command to Select Group of Employees from Table

This query selects all columns (represented by *) with surnames whose value is "Mundi".

To select specific columns (say, first_name, surname, and employee_number), instead of all columns, we specify the desired column names as illustrated below:

```
SELECT first_name, surname, employee_number FROM employee WHERE
surname="Mundi";
```

Fig. 2.6: Command to Select Specific Group of Columns from Table

The result will be as follows:

Table 2.6: Result of the Selected Columns from Table

first_name	Surname	employee_number
Yashim	Mundi	0001
Takon	Mundi	0004

Rather than displaying the entire table, only the specified columns are displayed.

Supposing we want to insert an additional row into our employee table, we could write the following piece of code:

```
INSERT INTO employee
(employee_number, first_name, surname, date_of_birth)
VALUES ('0005', 'John', 'Doe', '05-05-1977');
```

Fig. 2.7: Command to Insert an Additional Row into Employee Table

To alter a specific row, say for instance to change the date_of_birth of employee_number 0002 to May 5 1991, we use the update command as follows:

```
UPDATE employees SET date_of_birth='05-05-1991' WHERE
employee_number='0002';
```

Fig. 2.8: Command to Update Employee Table

Finally, supposing we want to delete the 2 employees whose surnames are "Mundi", we would issue the following SQL command:

```
DELETE FROM employee WHERE surname='Mundi';
```

Fig. 2.9: Command to Delete from Employee Table**3.2.3 Data Control Language (DCL)**

This subset of SQL commands is used to control access to data stored in a database. Below is a table that summarises the DCL commands, their descriptions and their basic syntax:

Table 2.7: Data Control Language (DCL)

COMMAND	DESCRIPTION	SYNTAX
GRANT	Authorises one or more users to perform an operation or a set of operations within a database.	Grant operation1,operation2,..., operationn on tablename to user1,user2,..., usern
REVOKE	Eliminates the authorisation of one or more users to perform a database operation. It's the opposite of the grant command.	Revoke operation1,operation2,..., operationn on tablename from user1,user2,..., usern

DCL commands are used by the database administrator to determine the access/usage rights that database users possess. Depending on for instance, the rank of a user within an organisation, he/she may only be allowed read (select) privileges on tables directly related to his/her job.

Suppose for instance we want to limit a user named “jane” to select and update operations on the employee table, we would use the following command:

```
GRANT SELECT, UPDATE ON employee TO jane
```

Fig. 2.10: Command to Perform Multiple Operations on Employee Table

Provided the user,, Jane, had no other privileges, she would be restricted to performing only select and update commands on the employee table. She would be unable to for instance insert and delete employee records.

Supposing that we want to revoke all of Jane's privileges because she has gone on a leave of absence from the company, we would issue the following command:

```
REVOKE * ON employee FROM jane
```

Fig. 2.11: Command to Revoke All Jane from Perform Operation on Employee Table

Using the * symbol, we have successfully revoked ALL privileges assigned to the user "jane". Thus Jane will be unable to perform any operation on the employee table.

4.0 CONCLUSION

Though a relatively simple language to pick up, the power of SQL is immense. SQL commands can grow to become very complex depending on the specific operation required to be performed on the database. In particular, it is possible to select data spanning multiple tables, compare data across tables and even perform table and database backups using a combination of SQL commands. Furthermore, the specific implementations of SQL across vendors such as Microsoft and MySQL provide additional database management tools including mechanisms for Master-Slave database replication and clustering for additional redundancy in mission-critical environments. The reader is encouraged to explore more specific implementations of SQL to learn about the nuances of writing queries.

5.0 SUMMARY

In this unit, the reader is provided with an introduction to SQL. We start off by discussing a brief history of SQL and conclude by taking a look at the various classes of SQL commands, providing brief examples to demonstrate their usage for simple but common operations.

6.0 TUTOR-MARKED ASSIGNMENT

Write SQL commands to perform the following operations:

- i. Create a table called “student” with the following columns: student_id, first_name, last_name, department, phone_number. For each field, determine the most appropriate data type.
- ii. Populate the student table in i. above with 5 complete records.
- iii. Empty the table in ii.
- iv. Alter the structure of the table by adding a field for addresses to be stored.

7.0 REFERENCES/FURTHER READING

Itzik, Ben-Gan (2008). *Microsoft SQL Server T-SQL Fundamentals*.
Microsoft Press.

Tutorialspoint.com, *SQL Overview*.

Online]http://www.tutorialspoint.com/sql/sql-
Overview.htm

UNIT 3 INTEGRATED WEB APPLICATION

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 System Architecture
 - 3.1.1 Client-Side Architecture
 - 3.1.2 Server-Side Architecture
 - 3.1.3 Database Architecture
 - 3.2 System Code
 - 3.2.1 Database Code
 - 3.2.2 Client-Side Code
 - 3.2.3 Server-Side Code
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Reading

1.0 INTRODUCTION

Learning individual web technologies is of little use if they cannot be tied together by a developer to build useful web applications. In the unit, we will demonstrate how to tie the three parts of a web application – client-side, server-side and database – together to build an address book which is capable of storing and retrieving contacts.

Our tools of choice will be HTML (for the client side), PHP (for the server-side) and MySQL (for the database). The choice of the aforementioned technologies is based on the facts that not only are they popular and very powerful, but are also free and open-source. On the Windows platform for instance, XAMPP –freely available for download at www.apachefriends.org/en/xampp.html -- comes bundled with PHP and MySQL and installation is simplified to a series of mouse clicks.

Before we dive into writing code for the application, we will take a look at the basic architecture of the system outlining the requirements of each part of the application.

2.0 OBJECTIVES

At the end of the unit, you should be able to:

- design small web applications, outlining the requirements and structures of each of the necessary components
- write the code corresponding to the functionality of each of the parts of the web application.

3.0 MAIN CONTENT

3.1 System Architecture

3.1.1 Client-Side Architecture

The client-side will be made up of two HTML forms:

- A form to enter contact information for storage in the database. For simplicity, we will refer to this form as the entry form. The entry form will be a static HTML form with fields for the following data items:
 - a. First Name
 - b. Last Name
 - c. Address
 - d. Phone Number
 - e. Email
 - f. Birthday
- A form to search the address book for contact information. For simplicity, we will refer to this form as the search form. The search form will be also be a static HTML form but it will contain one field in which a user can enter either a name, email address or phone number to search the address book for matching contact data.

The processing of both forms will be handled by separate server-side scripts.

3.1.2 Server-Side Architecture

The server-side will be made up of two PHP scripts which will handle the exchange of data between the client-side and the database.

The first script (which for convenience we will call `entry.php`) will be responsible for taking the data from the entry formant inserting it into the database.

The second script (which for convenience we will call `search.php`) will be responsible for querying the database for contact information based on search criteria supplied by the user. If matches are found, the script will display a tabular representation of the matching results.

3.1.3 Database Architecture

The MySQL database will serve as the backend of the application. It will consist of a single table which for convenience we will call `contacts`. It will be made up of 6 fields as follows:

- `contact_id` – A unique identifier for contact data. It will be of the `INT` data type.
- `first_name` – Stores the first name of each contact. It will be of the `VARCHAR` datatype with a length of 50 characters.
- `last_name` – Stores the last name of each contact. `VARCHAR (50)`.
- `address` – Stores the address of each contact. It will be of the `TEXT` data type since it could grow quite long.
- `phone_number` – Stores the phone number of each contact. `VARCHAR (20)`.
- `email` – stores the email address of each contact. `VARCHAR (100)`.
- `birthday` – stores the date of birth of each contact. It will be of the `DATE` data type.

3.2 System Code

Because we need the database to exist before we can make any attempt to connect to it, we begin with the database code and then move on to the client-side and finally the server-side code.

3.2.1 Database Code

We start by creating a database as follows:

```
CREATE database contacts;
```

Fig. 3.1: Command Creating Database

Next we select our newly created database and then create the contacts table within it as follows:

```
use contacts;

CREATE TABLE `contact` (
  `contact_id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(50) NOT NULL,
  `last_name` varchar(50) NOT NULL,
  `address` text NOT NULL,
  `phone_number` varchar(20) NOT NULL,
  `email` varchar(100) NOT NULL,
  `birthday` date,
  PRIMARY KEY (`contact_id`)
)
```

Fig. 3.2: Command Creating Table for the Database

The `contact_id` field is specified as “NOT NULL” to indicate that it must have a value for each record i.e. an empty field value is not permitted. Furthermore, it is specified as “AUTO_INCREMENT” to ensure that it generates the next `contact_id` value automatically. This means that the programmer doesn’t have to remember what the last `contact_id` is when he inserts a record into the table because the table will automatically generate the next `contact_id`.

A screenshot of the table is provided below:

#	Column	Type	Collation	Attributes	Null	Default	Extra	Action
1	contact_id	int(11)			No	None	AUTO_INCREMENT	Change Drop More
2	first_name	varchar(50)	latin1_swedish_ci		No	None		Change Drop More
3	last_name	varchar(50)	latin1_swedish_ci		No	None		Change Drop More
4	address	text	latin1_swedish_ci		No	None		Change Drop More
5	phone_number	varchar(20)	latin1_swedish_ci		No	None		Change Drop More
6	email	varchar(100)	latin1_swedish_ci		No	None		Change Drop More
7	birthday	date			Yes	NULL		Change Drop More

Fig. 3.3: Outlook of the Table Created Above

3.2.2 Client-side Code

The client-side code is made up of 2 HTML files:

- The first file will contain a form for entry of contact details. For the purpose of this unit, we will name the file *entry.html*. The contents of *entry.html* are listed below:

```
<html><!--entry.html --><head><title>Entry form</title></head><body>
<form method="post" action="entry.php">
<p>First Name: <input type="text" name="first_name" size="30" /></p>
<p>Last Name: <input type="text" name="last_name" size="30" /></p>
<p>Address: <textarea name="address" rows="5"
cols="30"></textarea></p>
<p>Phone Number: <input type="text" name="phone_number" size="15"
/></p>
<p>Email: <input type="text" name="email" size="30" /></p>
<p>Birthday(YYYY-MM-DD): <input type="text" name="birthday"
size="12" /></p>
<p><input type="submit" name="submit" value="Save Contact" /></p>
```

Fig. 3.4: Form for Entry of Contact Details

The form captures the contact details of each contact and when the “save contact” button is pressed, the form entries are processed by *entry.php*.

- The second file will contain a form with a single text box for searching the address book. For the purpose of this unit, we will name the file *search.html*. The contents of *search.html* are listed below:

```
<html><!--search.html --><head><title>Search form</title></head><body>
<form method="post" action="search.php">
<p>First Name / Last Name / Phone Number / Email: <input type="text"
name="search_field" size="30" /></p>
<p><input type="submit" name="submit" value="Search" /></p>
</form></body></html>
```

3.2.3 Server-side Code

As mentioned in the section on server-side architecture, the server-side is made up of 2 files: entry.php and search.php:

entry.php captures the data filled in the entry form, connects to the database, inserts the data into the database and displays a confirmation message if the data is successfully saved. The code for the file entry.php is given below:

```
<?php

//Capture the entries from the entry.html form in simple PHP variables.

$first_name=$_POST['first_name'];
$last_name=$_POST['last_name'];
$address=$_POST['address'];
$phone_number=$_POST['phone_number'];
$email=$_POST['email'];
$birthday=$_POST['birthday'];

//Add contacts to database

mysql_connect("localhost","root",""); // Connects the user to the
database mysql_select_db("contacts"); //Selects the contacts
database

$insert_query="INSERT INTO contact(contact_id
,first_name,last_name,address,phone_number,email,birthday)
VALUES (NULL , '". $first_name."', '". $last_name."', '". $address."',
' ". $phone_number."', ' ". $email."', ' ". $birthday."")";

mysql_query($insert_query); // Execute the insert query

//Display Confirmation message

echo "Contact successfully added! View the details below:<br />";
echo "<br>First Name: " . $first_name;
echo "<br>Last Name: " . $last_name;
echo "<br>Address: " . $address;
echo "<br>Phone Number: " . $phone_number;
echo "<br>Email: " . $email;
echo "<br>Birthday: " . $birthday;

?>
```

Fig. 3.6: Form for Capturing Data Filled in the Entry Form

Firefox | Entry form | HTML comment tag

localhost/addressbook/entry.html

Disable Cookies CSS Forms Images Information Miscellaneous Outline

First Name:

Last Name:

Address:

Phone Number:

Email:

Birthday(YYYY-MM-DD):

Fig. 3.7: Outlook of the Entry Form

When the form (entry.html) is filled as displayed above and the “save contact” button is pressed, entry.php performs the specified form processing yielding the screenshot displayed below:

Firefox | http://localhost/a...ressbook/entry.php | HTML comment tag | localhost / localhost

localhost/addressbook/entry.php

Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools

Contact successfully added! View the details below:

First Name: John
 Last Name: Oluwatobi
 Address: Plot D6, 5th Avenue, Parkview Estate, Lagos.
 Phone Number: 0813221233
 Email: johntobi@yahoo.com
 Birthday: 1988-12-12

Fig. 3.8: Outlook of the Form Processed

search.php captures the search item (either a name, email address or phone number) supplied by the user, queries the database with the search item and returns contacts that match the search. The code is given below with comments indicating the functionality handled by the different parts:


```

<?php $search_field=$_POST['search_field'];
mysql_connect("localhost","root",""); // Connects the user to the database
mysql_select_db("contacts"); //Selects the database

$select_query="SELECT * FROM contact WHERE first_name LIKE ".$search_field." OR last_name
LIKE ".$search_field." OR phone_number LIKE ".$search_field." OR email LIKE ".$search_field."";

$result=mysql_query($select_query); // Executes the select query and stores the result in a
variable called $result

echo "Results:<br>";
echo "<table border='1'><tr><td>First Name</td><td>Last Name</td><td>Address</td><td>Phone
Number</td><td>Email</td><td>Birthday</td></tr>"; // Displays a table with headings for First
Name, Last Name, Address, Phone Number, Email and Birthday

while(list($contact_id,$first_name,$last_name,$address,$phone_number,$email,$birthday)=mysql
_fetch_row($result))

// Fetches the required fields from the result variable and displays them in a table one row after
the other

        {

echo"<tr><td>".$first_name."</td>";
echo "<td>".$last_name."</td>";
echo "<td>".$address."</td>";
echo "<td>".$phone_number."</td>";
echo "<td>".$email."</td>";
echo "<td>".$birthday."</td></tr>";

echo "</table>";

```

Fig. 3.9: Commands that Create Search Form

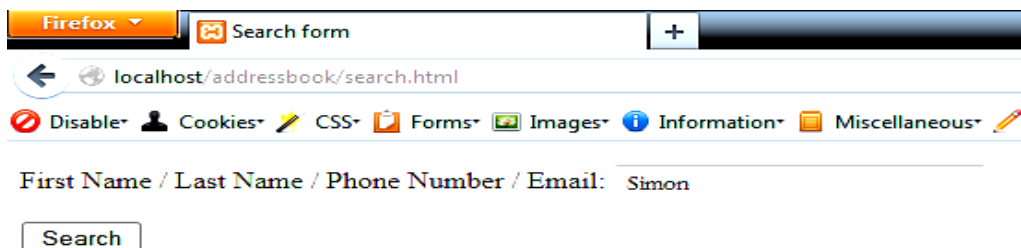


Fig. 3.10: Outlook of the Created Search Form

Results:

First Name	Last Name	Address	Phone Number	Email	Birthday
Simon	James	No 15 Ibeju-Lekki Road, Lagos	08012314412	simonjames@yahoo.com	1929-12-12
Simon	Dung	Plot 24, Ibo Street, Obalende, Lagos.	08092210123	simondung@yahoo.com	1989-10-12

Fig. 3.11: Outlook/Result of the Search Items

When the search form is filled as displayed above and the search button is clicked, the search executes the necessary queries and displays results as illustrated in the screenshot below

4.0 CONCLUSION

The functionality of the address book application we have illustrated here is far from complete. Additional operations that should be integrated into the application include the ability to delete and update contact information as well as import contacts (in bulk) from external data sources.

Notwithstanding, the building blocks necessary to implement additional functions are very similar to what we've looked at in the unit and the reader is encouraged to explore an expansion of the application with any feature(s) he/she thinks reasonable.

5.0 SUMMARY

In this unit, we took a look at how to integrate the various parts of a web application together to build a useful application to store and retrieve database contacts. We highlighted the roles of the client-side, server-side and database (backend) and proceeded to define specific architectural components in each case. We concluded the unit by taking a look at the code required by the various parts of the web application.

6.0 TUTOR-MARKED ASSIGNMENT

Expand the address book application in this unit to include functionality for updating contact information as well as deleting contact entries from the address book.

7.0 REFERENCES/FURTHER READING

Castagnetto, J. *et al.* (1994). *Professional PHP Programming*. Wrox Press: USA.

Valade, J. (2004). *PHP/MYSQL for Dummies*. (2nded.). Wiley Publishing Inc: USA.