NATIONAL OPEN UNIVERSITY OF NIGERIA

SCHOOL OF SCIENCE AND TECHNLOGY

COURSE CODE: CIT 853

COURSE TITLE: INTERNET CONCEPTS AND WEB DESIGN

COURSE GUIDE

CIT 853
INTERNET CONCEPTS AND WEB DESIGN

Course Team Mr. O. J. Oyelade (Course Developer/Writer) – Covenant University, Ota

NATIONAL OPEN UNIVERSITY OF NIGERIA

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

Abuja Office
No. 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 2013

ISBN: 978-058-482-X

All Rights Reserved

Printed by:

CONTENTS **PAGE**

INTRODUCTION

Internet Concept and Web Design is a three- credit unit degree course available to all students offering the Bachelor of Science (B.Sc.) in Computer Science (CIT).

Internet Concept and Web Design is a specialised area in the field of Computer Science. This course discusses the topics which include: Overview of the Internet, Concepts of website design and how to evaluate websites; JavaScript and VBScript; extensible Markup language and introduction to Micromedia Dreamweaver. Students are expected to complete extensive Internet and Web Design oriented projects to demonstrate mastery of the skills discussed in this course.

WHAT YOU WILL LEARN IN THIS COURSE

This course guide tells you briefly what the course is all about, what course materials you will be using and how you can work your way with these materials. In addition, it advocates some general guidelines for the amount of time you are likely to spend on each unit of the course in order to complete it successfully.

It gives you guidance in respect of your Tutor-Marked Assignment which will be made available in the assignment file. There will be regular tutorials classes that are related to the course. It is advisable for you to attend these tutorial sessions. The course will prepare you for the challenges you will meet in the field of Computer Science.

COURSE AIM

The major aim of the course is to provide you with an understanding of the introduction to Web design and Internet programming in the field of Computer Science and also to expose you to practical application of the Internet and Web Design in real life.

COURSE OBJECTIVES

To achieve this aim, the course has a set of objectives. Each unit has specific objectives which are included at the beginning of the unit. You should read these objectives carefully before you study the unit. You may wish to refer to them during your study to check on your progress. You should always look at the unit objectives after completion of each unit. By doing so, you would have followed the instruction in the unit.

Below are the comprehensive objectives of the course as a whole. By meeting these objectives, you should have achieved the aim of the course as a whole. In addition to the aim above, this course sets out to achieve some objectives. Therefore, after going through the course, you should be able to:

- explain the overview of the Internet
- describe the structure of web applications and its architecture
- explain the concept of client/server computing
- explain the fundamental of web design
- adding cascading style sheet to HTML
- format HTML with CSS font and text properties
- explain the concept of javascript and vbscript
- explain the fundamental of extensible markup language (XML)
- explain the concept of document type definition and XML schema
- explain the basic concept of micromedia dream weaver.

WORKING THROUGH THIS COURSE

To complete this course, you are required to read each study unit, read the textbooks and read other materials which may be provided by the National Open University of Nigeria. Each unit contains self-assessment exercises and at certain points in the course you will be required to submit assignments for assessment purposes. At the end of the course there is a final examination. The course should take you about a total of 17–18 weeks to complete. Below you will find listed all the components of the course, what you have to do and how you should allocate your time to each unit in order to complete the course on time and successfully.

This course entails that you spend a lot of time to read. I would advise that you avail yourself the opportunity of attending the tutorial sessions where you have the opportunity of comparing your knowledge with that of other people.

THE COURSE MATERIALS

The main components of the course are:

1. The Course Guide
2. Study Units
3. References/Further Reading
4. Assignments

5. Presentation Schedule

STUDY UNITS

The study units in this course are as follows:

Module 1 consists of two units. In unit 1, it addresses the overview of the Internet, that is, the definition, administration of the Internet, domain name and IP addresses, information routing on the Internet and application layer protocols. The second unit deals with the introduction to the Web. Module 2 consists of five units; Unit 1 deals with introduction to Web design where it discusses on the common gateway interface, hypertext markup language(HTML) and element of HTML. Unit 2 discusses on adding graphics to Web pages. Units 3, 4 and 5 deal with Cascading Style Sheet(CSS) where it discusses on the brief history of CSS, defining global style in CSS, how to insert CSS into HTML document, style sheet procedure rules, its properties, CSS box properties and classification properties. Module

3 consists of four units; unit 1 discusses on JavaScript such as types of JavaScript, variables in JavaScript, the operators and JavaScript control flow. Unit 2 deals with JavaScript and introduction to object oriented programming, while units 3 and 4 focus on visual basic script and how to control VBScript routines. Module 4 consists of five units; Unit 1 introduces the XML basics, unit 2 discusses on Document Type Definition (DTDs) where it focuses on creating a DTDs and how to validate an XML document with a DTD, unit 3 focuses on XML schema basics where it discusses on the purpose and power of XML schema, unit 4 discusses on XSLT basics and lastly unit 5 deals with the introduction to micromedia dreamweaver.

Each unit consists of one or two weeks' work and include an introduction, objectives, activities, conclusion, and summary, Tutor-Marked Assignments (TMA), references and further reading. In general, these activities test you on the materials you have just covered or require you to apply it in some way and thereby assist you to evaluate your progress and to reinforce your comprehension of the material. Together with TMAs, these activities will help you in achieving the stated learning objectives of the individual units and of course in general.

TEXTBOOKS AND REFERENCES

Alberto, L. & Indra, W. (2000). *Communications and Networking: Fundamental Concepts and Key Architecture*.. McGraw-Hill.

Brooks & David, R (2007). An Introduction to HTML and JavaScript for Scientists and Engineers.

Charles, F. G. & Paul, P.(1998). *The XML Handbook*.. Prentice Hall Computer Books.

David, F. (1998). *JavaScript*: *The Definitive Guide* (3rd ed.).. O'Reilly Media.

David, S. M. (2006). *Dreamweaver 8: The Missing Manual*.

Don, J. &Jeffery, H. (2006). *Advanced VBScript for Microsoft Windows Administrators*.. Microsoft Press.

Ed, Wilson (2006). “Microsoft VBScript: Step by Step”. Microsoft Press.

Elliotte, R. H. & Scott, M.W. (2001). “XML in a Nutshell”.

Janine, W. (2005). "Teach Yourself VISUALLY Macromedia Dreamweaver 8". Visual.

Larry, R.(2002). *Programming the Web using XHTML and JavaScript*. McGrawHill.

Michael, M. (2001). " HTML and XML for Beginners"

Richard, A. M.(2003). *Introduction to Networkin*. McGraw-Hill.

William, B. (2000). *Distributed Systems Networks*. .McGraw-Hill.

http://dev.opera.com/articles/view/4-the-web-standards-model-html-css-a/
http://www.corewebprogramming.com

ASSIGNMENT FILE

These are of two types: the self-assessment exercises and the tutor-marked assignment. The self-assessment exercises will enable you monitor your performance by yourself, while the tutor-marked assignment is a supervised assignment. The assignments take a certain percentage of your total score in this course. The tutor-marked assignments will be assessed by your tutor within a specified period. The examination at the end of this course will aim at determining the level of mastery of the subject matter. This course includes sixteen tutor-marked assignments and each must be done and submitted accordingly. Your best scores however, will be recorded for you. Be sure to send these assignments to your tutor before the deadline to avoid any penalty.

PRESENTATION SCHEDULE

Your course materials have important dates for the early and timely completion and submission of your TMAs and attending tutorials. You should remember that you are required to submit all your assignments by the stipulated time and date. You should guard against falling behind in your work.

ASSESSMENT

There are three aspects to the assessment of the course. First is made up of self-assessment exercises, second consists of the tutor-marked assignments and third is the written examination/end of course examination.

You are expected to do all the activities. In tackling the assignments, you are expected to apply information, knowledge and techniques you gathered during the course. The assignments must be submitted to your facilitator for formal assessment in accordance with the deadlines stated in the presentation schedule and assignment file. The work you submit to your tutor for assessment will count for 30% of your course work. At the end of the course you will need to sit for a final or end of course examination of about three hour duration. This examination will count for 70% of your total course work.

TUTOR-MARKED ASSIGNMENT

The TMA is a continuous assessment component of your course. It accounts for 30% of the total score. You will be given four (4) TMAs to answer. Three of these must be answered before you are allowed to sit for the end of your course examination. The TMAs would be given to you by your facilitator and returned after you have done the assignment. Assignment questions for the units in this course are contained in the assignment file. You will be able to complete your assignment from the information and material contained in your reading, references and study units. However, it is desirable in all degree level of education to demonstrate that you have read and researched more into your references, which will give you a wider view point and may provide you with a deeper understanding of the subject.

Make sure that each assignment reaches your facilitator on or before the deadline given in the presentation schedule and assignment file. If for any reason you cannot complete your work on time, contact your facilitator before the assignment is due to discuss the possibility of an extension. Extension will not be granted after the due date unless there exceptional circumstances.

FINAL EXAMINATION AND GRADING

The end of course examination for Internet Concepts and Web Design will be for about 3 hours and it has a value of 70% of the total course work. The examination will consist of questions, which will reflect the type of self-testing, practice exercise and tutor-marked assignment problems you have previously encountered. All areas of the course will be assessed.

Revise the whole course thoroughly before sitting for the examination. You might find it useful to review your self-test, TMAs and comments on them before the examination. The end of course examination covers information from all parts of the course.

COURSE MARKING SCHEME

Assignment	**Marks**
Assignment 1-4	Four assignments, best three marks of the four count at 10% each-30% of course marks
End of course examination	70% of overall course marks
Total	100% of course material

COURSE OVERVIEW

The course starts with an exploration of core protocols that underpin the basic client-server model of the World Wide Web, leading to consideration of the various technical standards and recommendations that determine the tools and techniques employed for the creation and deployment of digital information. An in-depth introduction to the Internet and the World Wide Web for more advanced studies in Web programming, Internet tools, and Web document publishing including HTML and Cascading Style Sheets, Internet design and communication protocols including: TCP/IP, FTP, HTTP, SMTP, telnet and the tools that use them. The course also provides a thorough grounding in the use of XHTML elements and attributes for the creation of static content and Cascading Style Sheets (CSS) style rules to control presentation. The course also examines the technical requirements for usability and accessibility and explores how different presentational structures have evolved to meet the requirements of specific applications.

Unit	Title of Work	Weeks Activity	Assessment (End of Unit)
	Course Guide	Week 1	
	Module1		
1	Overview of the Internet	Week 1	Assignment 1
2	Introduction to the web	Week 2	Assignment 2
	Module2		
1	Introduction to the web design	Week 3	Assignment 3
2	Adding graphics to web pages	Week 4	Assignment 4
3	Cascading style sheet (CSS)	Week 5	Assignment 5
4	Cascading style sheet properties	Week 6	Assignment 6
5	Cascading style sheet properties continue	Week 7	Assignment 7
	Module3		
1	JavaScript	Week 8	Assignment 8
2	JavaScript and Object Oriented Programming (OOP)	Week 9-10	Assignment 9
3	Visual basic Script	Week 11	Assignment 10
4	VbScript object and controlling VbScript routines	Week 12-13	Assignment 11
	Module4		
1	XML Basics	Week 14	Assignment 12
2	Document Type Définitions (DTDs)	Week 15	Assignment 13
3	XML Schema Basics	Week 16	Assignment 14
4	XSLT Basics	Week 17	Assignment 15
5	Micromedia Dreamweaver		Assignment 16
	Revision	Week 18	
	Examination	Week 19	
Total		19 weeks	

HOW TO GET THE MOST FROM THIS COURSE

Prepare for the study units so that you will be more likely to predict the organisation of the module. Check the course outline to see if the tutor has listed the topic or key ideas in the next module. If so, convert this information into questions, or structure your notebook according to the headings provided in the outline.

Using a combination of course website, a book, and electronic guides, you will study a range of techniques for creating web pages comprising text, images, image maps, hyperlinks, tables, frames, and forms. This will provide the knowledge and skills expected from you in this course.

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 enable 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 units 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.

Remember that your tutor's job is to assist you. When you need help, do not hesitate to call and ask your tutor to provide it.

1. Read this *Course Guide* thoroughly.
2. 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. Whatever method you chose to use, you should decide on it and write in your own dates for working on each unit. Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they lag behind in their course work.
3. Assemble the study materials. Information about what you need for a unit is given in the 'Overview' at the beginning of each unit. You will almost always need both the study unit you are working on and one of your set of books on your desk at the same time.
4. Work through the unit. The content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit you will be instructed to read sections from your set books or other articles. Use the unit to guide your reading.
5. Review the objectives for each study unit to confirm that you have achieved them. If you feel unsure about any of the objectives, review the study material or consult your tutor.
6. When you are confident that you have achieved a unit's objectives, you can then start on the next unit. Proceed unit by unit through the course and try to pace your study so that you keep yourself on schedule.
7. After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this *Course Guide*).

FACILITATORS/TUTORS AND TUTORIALS

There are 16 hours of tutorials provided in support of this course. You will be notified of the dates, times and location of these tutorials as well as the name and phone number of your facilitator, as soon as you are allocated a tutorial group.

Your facilitator will mark and comment on your assignments, keep a close watch on your progress and any difficulties you might face and provide assistance to you during the course. You are expected to mail your Tutor-Marked Assignment to your facilitator before the schedule date (at least two working days are required).They will be marked by your tutor and returned to you as soon as possible.

Do not delay to contact your facilitator by telephone or e-mail if you need assistance.

The following might be circumstances in which you would find assistance necessary, hence you would have to contact your facilitator if:

- you do not understand any part of the study or the assigned readings
- you have difficulty with the self-tests
- you have a question or problem with an assignment or with the grading of an assignment.

You should endeavour to attend tutorials. This is the only chance to have face to face contact with your course facilitator and to ask questions which are answered instantly. You can raise any problem encountered in the course of your study.

To gain much benefit from course tutorials prepare a question list before attending them. You will learn a lot from participating actively in discussions.

SUMMARY

Internet Concepts and Web Design is a course that intends to provide concept of the discipline and is also concerned with the application of it to real life. Upon completing this course, you will be equipped with the basic understanding of the Internet, introduction to Web Design, JavaScript and VBScript, basic knowledge of extensible Markup language, and introduction to Micromedia Dreamweaver.

I wish you success in this course and I hope that you will find it both interesting and useful.

MAIN COURSE

CONTENTS **PAGE**

MODULE1 THE INTERNET

UNIT 1 OVERVIEW OF THE INTERNET

CONTENTS

1.0 INTRODUCTION

The Internet came as a result of the crave for a robust, efficient, store and forward, data network based on packets (packet- switching) as against the circuit-switching (telephone network) previously in existence. The Internet was developed by Paul Baran and Donald Davis in 1962. This packet switching technology was first implemented in the US Defense Advanced Research Projects Agency (DARPA) as ARPANET, a large area network developed by the Advanced Research projects Agency in late 60s. By early 70s ARPANET had spanned the US continent and was extended to some parts of Europe by 1973 and later to the rest of the world.

By 1986, the US Science Foundation (NSF) initiated a network called NSFNET, which later became a major component of the Net. Similarly, other networks were developed throughout the US through which the rest of the world was connected to form a global network of systems and network called the Internet.

2.0 OBJECTIVES

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

- explain the Internet and its origin
- explain the administration of the Net
- describe the concepts of domain names and IP addresses
- describe the DoD reference model
- explain the various application layer protocols.

3.0 MAIN CONTENT

What is the Internet?

The Internet or simply the Net as it is often called, stands for interconnected network of networks on a global scale. It makes it possible for computers all over the world to, send and receive messages. It is an internet-work of several hosts and their networks together to form a larger network of global magnitude. It is therefore a global collection of computers and networks that connects millions of peoples, organisations, military, and government to a wide range of information resources through a common protocol to communicate among themselves.

Considering the global nature of the Internet, which involves several systems, running on different platforms, intersystem communication becomes an issue. This problem is solved using the transmission control protocol/Internet protocol (TCP/IP), which defines the rules for communication on the Net.

3.1 Administration of the Internet

To govern the activities of the Net, the Internet Activities Band (IAB) was formed in 1983, with membership from professionals and researchers with technical interest in the health and the evolution of the entire Internet system. The committee was saddled with the responsibility of coordinating design, engineering and management of the Net. Figure 1.1gives a diagrammatic representation of Internet organisation.

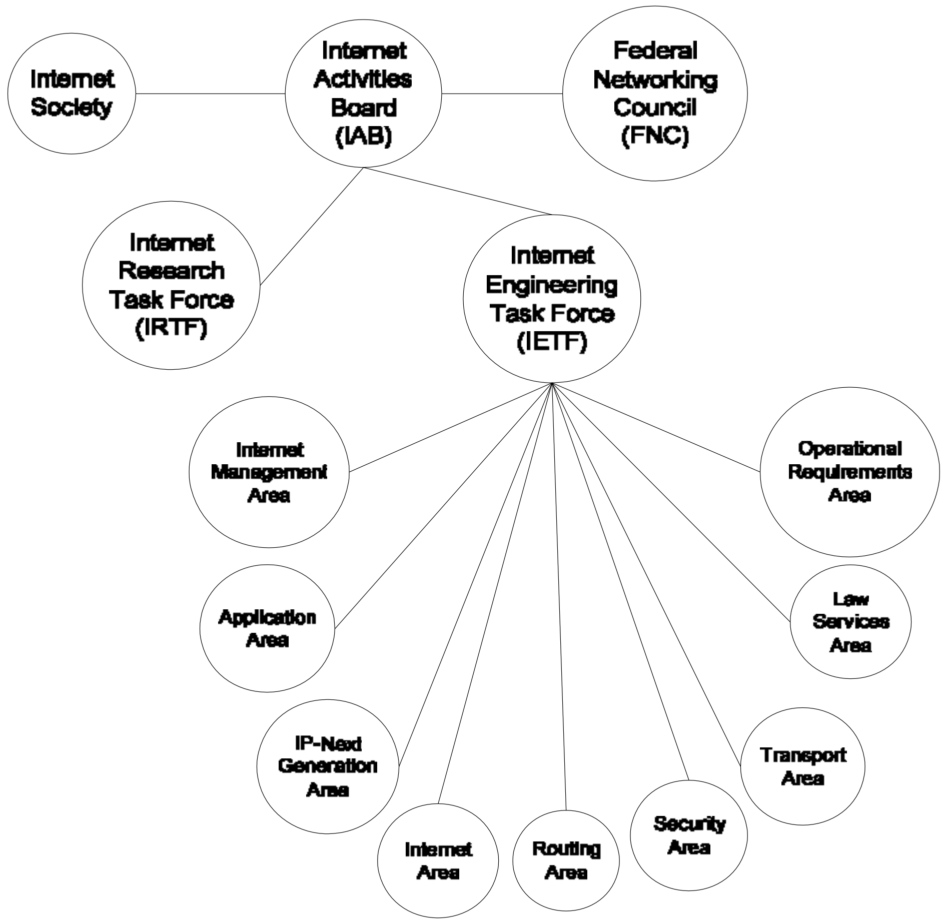

Internet Society
Internet Activities Board (IAB)
Federal Networking Council (FNC)
Internet Research Task Force (IRTF)
Internet Engineering Task Force (IETF)
Internet Management Area
Operational Requirements Area
Application Area
Law Services Area
IP-Next Generation Area
Transport Area
Internet Area
Routing Area
Security Area

Fig. 1.1: Organisation of the Net

The IAB has two task forces: The Internet Research Task Force (IRTF) and the Internet Engineering Task Force (IETF). Also, there are two organisations that liaise with IAB at the topmost level. They are the Internet Society (IS) and the Federal Networking Council (FNC). The IS draws its membership from the entire Internet community, while the FNS is for all agencies of the US Federal Government. Both IS and FNC are responsible for formulating policies and standards.

The IETF is responsible for formulating policies in areas of applications, Internet, IP-Next generations, Network management, Operational requirements, Routing, Security, Transport and User services.

3.2 Domain Names and IP Addresses

For the sake of our study, it is important to understand that all the entities and inter-mediaries of the Net are given unique names for easy identification and delivery of messages. Thus, all the hosts/nodes on the Internet are identified through a unique identifier called domain names.

NOTE: Hosts are the organisation who owns a particular network while nodes are the various computers connected the Internet via the host.

The major parties involved in information dissemination are the users (people) and the systems (IT equipment). Therefore, domain names must be convenient and informative for all the parties to operate effectively. Users prefer the alphabetic naming conventions to recognise the hosts, while the computers or systems use the IP addresses.

3.2.1 Domain Names

Generally, domain names are composed of 3 or 4 parts namely: Hostname, Organisation, Sub-domains (optional) and Country name.

e.g:
Hostname.Organisationname.Type of categorisation.Countryname
OR
Hostname.Domain.Top-level domain.International top-level domain.

Where the top-level domains include:

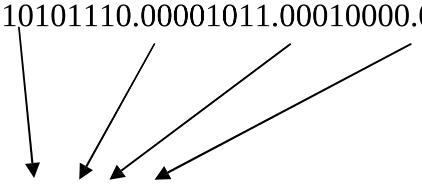
- .com for commercial
- .mil for military
- .edu for education
- .org for organisation
- .net for information services/network
- etc.
- and the international top-level domain use a 2-letter country code such as:
    - .ng for Nigeria
    - us for United States
    - .uk for United Kingdom
    - etc.

e.g:
1. www.openuniversity.edu.ng
2. www.openuniversity.ng
3. www.google.com
4. www.cs.ac.uk

3.2.2 IP Addresses

IP addresses are unique four octet numbers expressed either as binary dotted or decimal dotted. e.g.

10101110.00001011.00010000.01000001 Binary-dotted



179.11.16.65 Decimal-dotted

To convert IP address from binary to decimal form, we convert each of the four 8-bits numbers in each octet according to the table below:

Table 1.1

Decimal Value	128	64	32	16	8	4	2	1	Decimal
Octet Value (1)	1	0	1	1	0	0	1	1	128+32+16+2+1=179
Octet Value (2)	0	0	0	0	1	0	1	1	8+2+1=11
Octet Value (3)	0	0	0	1	0	0	0	0	16
Octet Value (4)	0	1	0	0	0	0	0	1	64+1=65

3.2.3 Address Classes

There are five different classes of address designed to meet the needs of different organisations. The various classes are given as A, B, C, D and E. The classes are distinguished from each other by the decimal notation of the first octet. The class range is presented in the table below:

Table 1.2

Class	First Octet	Net-ID	Default Subnet mask	Availability
A	1-126	First Octet	255.0.0.0	Available
B	128-191	First 2 Octet	255.255.0.0	Available
C	192-223	First 3 Octet	255.255.255.0	Available
D	224-239	N/A		Reserved for multicasting
E	240-255	N/A		Reserved

NB
127 is reserved for loopback (127.0.0.1) for internal testing on the local machine. Web applications can be tested on the local machines using the loopback (localhost) before deployment to the web.

Each system on a network or Internet is assigned a unique IP address by which it is identified. Furthermore, for easy identification and operations of the entire system, the domain name service (DNS), maintains a database of computer names and IP addresses corresponding to each of them. That is, the DNS is responsible for translating domain names to IP addresses and vice versa as occasions demand between the users and the systems.

For proper organisation and assignment of domain names and IP addresses, the Internet Corporation for Assigned Names and Numbers (ICANN), is saddled with the responsibility of managing and assigning names and IP addresses. ICANN is a special organisation with a charter from the US government to oversee the registration of names on the Internet alongside some domain name Registrars such as www.netsol.com and www.registrar.com to manage the process.

SELF-ASSESSMENT EXERCISE

Briefly explain the importance of Domain name and IP address.

3.3 Information Routing on the Internet

Data passage within an internet-work is via a router. That is, with the unique IP addresses assigned each of the systems, a packet of data is transmitted from one node to another through the router. Among the various network interconnecting devices such as bridge, switch, hub, repeater, gateway and router, the routers is the most ideal because of its capability to handle complex situations like the Internet, although it is the most expensive of them.

The router maintains a routing table, which contains the IP address of all the adjacent nodes, the various subnets of the Internet. Similarly, it has the capability of routing packets of data through the shortest route using a number of routing algorithms. These make it a preferred candidate for a complex network of great magnitude like the Internet.

3.3.1 The Internet Model

This model predates the open standard interconnection (OSI) model. It dates back to the ARPANET (the origin of the Internet) and hence, often referred to as the department of defense (DoD) model. The OSI model is composed of a seven-layered architecture along which network communications are segmented. Each layer covers a specific type of network activities, equipment or protocols. The layers are: Application, Presentation, Session, Transport, Network, Data Link, and Physical.

The DoD model is a four-layered architecture that does not map the OSI layer perfectly but can be relayed as follows:

OSI		DoD
Application Layer		Process/ Application Layer
Presentation Layer		
Session Layer		
Transport Layer		Host-to-Host Layer
Network Layer		Internetwork Layer
Data-Link Layer		Network Access Layer
Physical Layer		

Fig. 1.2: ISO and DoD Reference Model

Layer 4 Process/Application Layer
This layer combines the functionality of the topmost three layers of the OSI model. That is, application, presentation and session. The protocols include: e-mail, telnet, network management and directory services (NFS).

Layer 3 Host-to-Host Layer
This layer is the equivalent of the transport layer of the OSI model. It is responsible for end-to-end data integrity. The two protocols here are transmission control protocol (TCP) and the user datagram protocol (UDP). TCP offers reliable services and full duplex connections, while the UDP provides unreliable services that enhance throughout when error connection is not involved.

Layer 2 Internet Layer
This Layer corresponds to the network layer of the OSI model. Thus, it is responsible for routing packets within the internetwork. Gateways and routers are used for that purpose.

The TCP/IP protocol at this layer is the Internet protocol (IP) which operates a system of logical host addresses called IP addresses.

Layer 1 Network Access Layer

This combines the functionalities of both the physical and data link layers of the OSI model. It is responsible for exchange of data between a host and the network as well as delivery of data between two devices on the same network.

Communication between two systems through DoD model is as illustrated in figure 1.3 below:

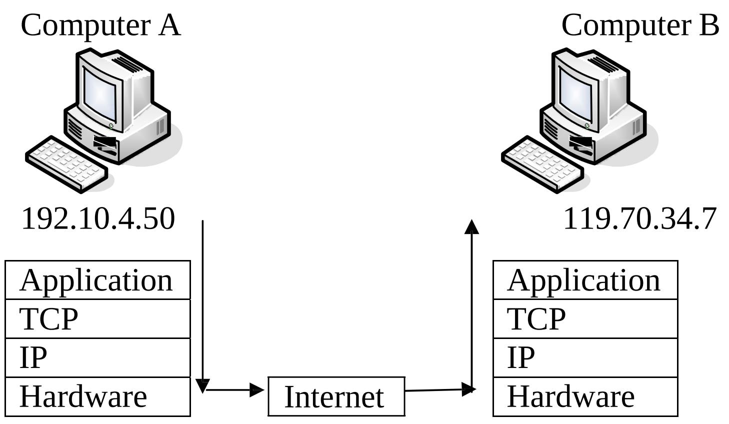


Fig. 1.3: Packet Transmission through the Net

Communication between two systems over the Internet is described as follows:

- The packet of data is sent down the four layers of the DoD model from application, through TCP, IP to Hardware (network cable, NIC) from where it is sent to the Internet and delivered to the destination system via the hardware (network cable, NIC), through IP, TCP and application.
- However, as the packet travels from one layer to another it wrapped into an “envelope” with the necessary information to the next layer. When received, it is unwrapped and rewrapped for the next lower layer. The procedure is repeated down the layers of the sending system and up the layers of the receiving system until it gets to the specified IP address. The process of wrapping and unwrapping packets of data is also known as encapsulation and decapsulation respectively.

3.4 Application Layer Protocols

1. HTTP – This is known as hypertext transfer protocol. It is a connectionless text-based protocol through which clients (web browsers) send requests to the web server for certain web pages and images to be displayed. After servicing this request, the connection between the client and the server across the Net is disconnect. However, for another request from the client to be serviced, another connection is sought.
2. Telnet – This is an application layer protocol that enables users to execute terminal sessions with remote hosts. That is, it allows a user to login to another host at remote location.
3. FTP – This stands for file transfer protocol. It enables transfer of files between two hosts that are at remote locations to each other. It performs basic file transfer between hosts.
4. SMTP – This stands for simple mail transfer protocol. This is a protocol that is used for exchanging electronic mail. It is used for basic message delivery.
5. SNMP – This is known as simple network management protocol. It is a protocol that is used to manage the network. It is used to collect information from the connected devices on the network for management purposes.

SELF-ASSESSMENT EXERCISE

i. Mention six network interconnecting devices.
ii. Mention the device most appropriate for transmitting information over the Internet and why?
iii. Briefly explain its mode of operation.
iv. Describe the OSI model with reference to the DOD model.

4.0 CONCLUSION

The Internet simply stands for interconnected network of networks on a global scale that makes it possible for computers all over the world to send and receive messages. It is an internet-work of several hosts and their networks together to form a larger network of global magnitude. It is a global collection of computers and networks that connects millions of peoples, organisations, military, and government to a wide range of information resources through a common protocol to communicate among themselves.

5.0 SUMMARY

In this unit, we have learnt the:

- meaning of the Internet and its origin
- administration of the Net
- concepts of domain names and IP addresses
- DoD reference model and
- various application layer protocols.

6.0 TUTOR-MARKED ASSIGNMENT

i. Discuss the origin of the Internet
ii. Explain briefly the concept of Client-Server Computing in terms of Layers and Tiers
iii. Explain briefly the application layer protocols.

7.0 REFERENCES/FURTHER READING

Alberto, L. & Indra, W. (2000). Communications and Networking: Fundamental Concepts and Key Architecture. McGraw-Hill.

Richard, A. M. (2003). Introduction to Networking. McGraw-Hill.

William, B. (2000). Distributed Systems Networks. McGraw-Hill.

UNIT 2 INTRODUCTION TO THE WEB

CONTENTS

1.0 INTRODUCTION

The World Wide Web popularly called the “web” is a global collection of interconnected documents on the Internet. Note that the web itself is not the internet. The Web is a part of the Internet that uses the Hypertext Transfer Protocol (HTTP) to display hypertext and images in a graphical environment. The World Wide Web is a system of internet servers that use **HTTP** (Hypertext Transfer Protocol) to transfer documents formatted in **HTML** (Hypertext Mark-up Language).

2.0 OBJECTIVES

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

- explain the structure of web application
- identify some web servers available
- explain the concepts of 3-tier and N-tier architecture
- explain the concepts of layers and tiers architecture.

3.0 MAIN CONTENT

The Structure of the Web Applications (WEBAPPS)

Webapps are applications that are accessed with a web browser over a network such as the Internet or an intranet. Every Webapps is divided into three major paths: the web browser, web server, and database server. This shall be discussed in detailed at section 2.2. They are popular because of the ubiquity of the browser as a client (thin client).

Similarly, the popularity is equally due to the possibility of updating and maintaining the application without necessarily distributing and installing it on every available client. Webapps are used to implement webmail, online retail sales, online auctions, discussion boards and weblogs among others.

Web developers often use client-side scripting to add functionality to the webapps by creating an interactive site that does not require page reloading. Webapps generate a series of web pages dynamically in a standard format supported by common browsers such as the hypertext markup language (HTML).

3.1 Basic WEAPPS Architecture

The web works based on the client/server Architecture. That is both the server (web server or middleware) and the client (web browser) applications are responsible for some sort of processing. Web application is structured as a 3-tier application. That is web browser constitutes the first tier, a middleware engine using some dynamic web content technology such as: common gateway interface (CGI), hypertext preprocessor (PHP), java servlets or java server page (JSP), active server pages (ASP) constitute the middle-tier and the database which we earlier called database server being the third tier. Figure 2.1 depicts the structure of web application with example of tools at work in each tier.

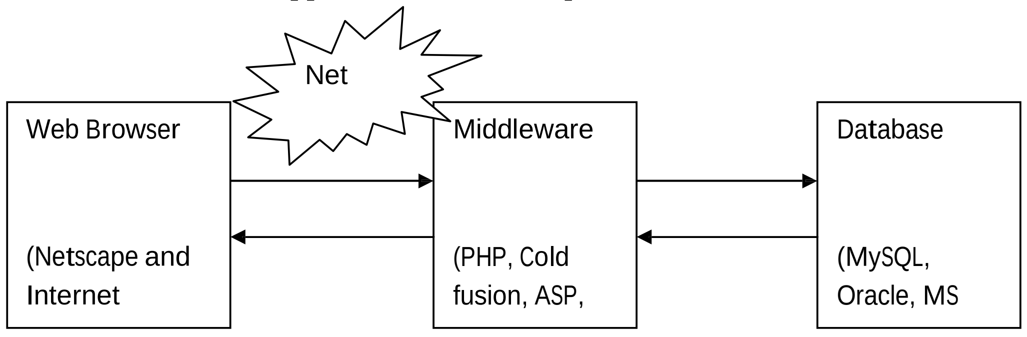


Fig. 2.1: Structure of Web Application

3.1.1 The Web Browser (Client)

The web browser constitutes the client. It is a software application that enables a user to display and interact with text, images and other information that are located on the web page the web or a local area network. Browsers are used for quick and easy access to information contained in the web pages or at some websites by traversing some links. Similarly, browsers can be used to access information on web servers. Examples of web browsers are MS Internet Explorer, Mozilla Firefox, Apple Safari, Netscape and Opera.

Web browser works by communicating with web servers using the hypertext transfer protocol (HTTP). HTTP allows web browsers to submit information to web servers as well as fetch web pages from them. The primary language of browsers is the HTML, which consists of tags that are used to describe a web page. HTML will be discussed in detailed in module 2. Most browsers also have some level of support for scripting languages like JavaScript and markup languages like extensible markup language (XML).

3.1.2 The Web Server (Middleware)

All Web transactions take place on the servers. The web server is responsible for communicating with the browser while the database server is responsible for storing the required information. The web server takes all requests from the clients, responds to the requests and serves the appropriate web pages back to the clients.

Examples of web servers
There are several web servers but the most prominent of them are Apache HTTP server and the Microsoft's Internet Information Services (IIS)

a Internet Information Services (IIS)
This is a major component of the Microsoft Server operating system, particularly a component of its active server pages (ASPs). IIS is recommended if both the middleware (ASP) and the database Server (SQL Server) are Microsoft products. IIS is the world's second most popular Web server.

b Apache HTTP Server
This is most popular web server. It is a free software/opensource. Apache runs on Unix, Linux, MS Windows, Novell Netware and some other platforms.

Apache serves over 68% of websites and serves both static and dynamic contents on the web in a very reliable and secure manner. It offers server-side programming language support for authentication schemes. Some common web servers are presented in the table below.
Table 2.1 Web Servers

The middleware is composed of languages such as PHP, ASP, ColdFusion, JSP and Perl. These languages work with web servers to interpret requests from clients, process the requests and interact with other programs that may be needed to fulfill the transactions and indicate to the web server the actual page to serve the client.

The middle-tier may be multi-tiered. That is, it can be composed of several other servers with designated responsibilities, hence the over-all architecture is said to be N-tier. A fundamental rule in 3-tier architecture is that the client has no direct line of communication with the data tier. That is, all communications are routed through the middleware tier.

3.1.3 Database Server

This is a program that provides database services to other computer programs or computers. Database Management Systems (DBMS) provide functionality to database servers. They are responsible for storing, retrieving and manipulating the data in the database or other repositories. Some popular DBMSs include: Oracle, Sybase, Informix, SQL server, Db2 and Interbase.

3.2 Client/Server Computing

This section will discuss the possible arrangement of web components in manners suitable for distributed computing. In this section, we shall also understand the client/server inter-relationship

There is a separation of functionality in client/server technology. The client (front–end) does data presentation and or processing, while the server (back-end) does storage, security and major data processing.

The client/server inter-relationship is given in terms of **Layers** and **Tiers**.

3.2.1 Layers

Layering describes the division of labour with the application codes resident on a single machine. That is both the client application, the server application, and the database sever is placed on different folders in the same machine. Layers involve code modules placed in different folders or directories on the client/server. The client-side code can be structured into 0-3 layer(s) of application code, while the server-side code can be structured into 1-3 layers of application code. It fosters code re-usability, security and convenience. It is a good software design.

The client-side can be categorised into

3.2.1.1 Thin Client

That is, a client with a zero code layer that has no custom application code running on it. The server holds all the custom application codes. This is shown in figure 2.2 below:

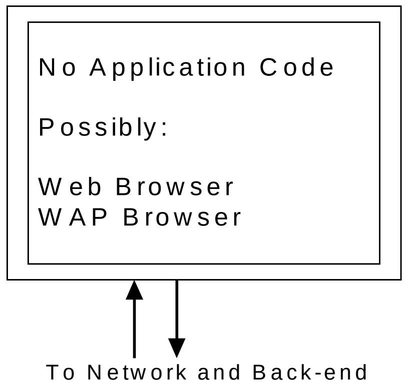


Fig. 2.2: Thin Client Architecture

3.2.1.2 Fat Client

This is a Client with 1 to 3 application code running on it. The three layers are:

a. Layer 1 Presentation Layer – The layer that interacts closely with the user.
b. Layer 2 Business layer – The layer that handles the business logic of the code.
c. Layer 3 Data access Layer – The layer that handles communication with database or data source.

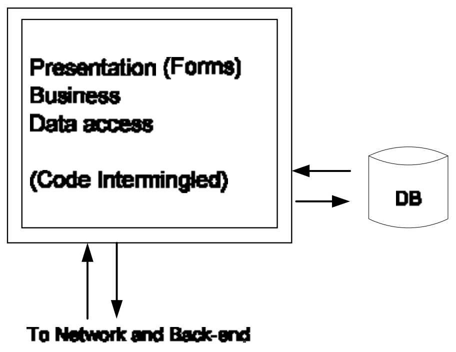


Fig. 2.3: Fat Client (Layer 1)

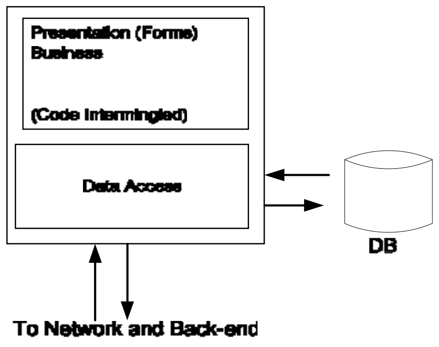


Fig. 2.4: Fat Client (Layer 2)

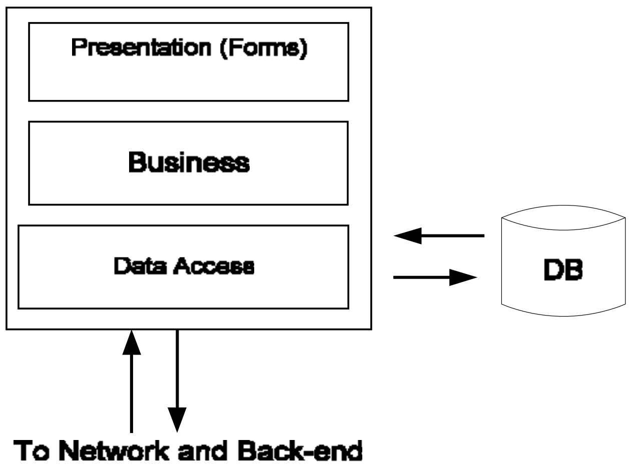


Fig. 2.5: Fat Client (Layer 3)

It is possible to have more than three layers on either side – But it is not encouraged for unwieldiness and difficulty to manage.

SELF-ASSESSMENT EXERCISE

i. Write short note on the following: Web browser, web server and database server,
ii. Distinguish between thin client and fat client.

3.2.2 Tiers

The Tiers describes the division of labour of application code on multiple machines. The code modules are placed on different machines in a distributed server environment. In a layered architecture, the various modules are resident on one single machine. That is, the layers are simply different folder of applications on one single computer.

However, in a tiered architecture, the codes are separated into three tiers (modules) namely:

a. Tier 1 - Presentation tier (PT), it interacts closely with the user.
b. Tier 2 - Application tier (AT), it holds the business logic and the data access logic.
c. Tier 3 - Database tier (DT), houses database or data source.

Each tier may be composed of several servers, e.g. in a large-scale web application environment, the architecture could be composed of:

- Large number of inexpensive servers (PT)
- Small number of application servers (AT)
- Few (2) number of clustered db servers (DT)

The architecture can be scaled horizontally or vertically. The horizontal scaling or scaling out involves the ability to add more servers. Vertical scaling or scaling up involves the ability to add more powerful servers.

In large scale distributed web applications, tiers are bounded by firewalls. One before (in front) presentation tier, another firewall before (in front) application tier. That is, the presentation is sandwiched between firewalls, generally termed demilitarised zone (DMZ).

Thus, the application and database tiers are shielded behind the second firewall termed the Internet zone (IZ).

Tiering fosters security and enables larger establishments to shield their internal systems from attacks. Without tiering, it is difficult securing internal systems.

Tiers -Generally refers to server architectures and not clients.

3.2.2.1 Clients

This includes virtually all devices such as: cell phones, research in motion (RIM) devices, PDAs, tablet PCs, notebooks and PCs.

a Thin Clients

- No application code, but relies on the server
- Uses web and WAP browsers to display
- Web (HTML)
- WAP (WML)
- It is easier to maintain and support since no application code or data is in them.
- It must be in constant communication with the server.

b Fat Clients

- It has 1-3 layers of application code
- It can operate for some time independent of the server
- It is useful in a situation where communication between client and server is not guaranteed
- It can store information in local database before sending to the server
- Client can work independent of the server

3.2.2.2 One-Tier Architecture

The three code layers exist on a single server

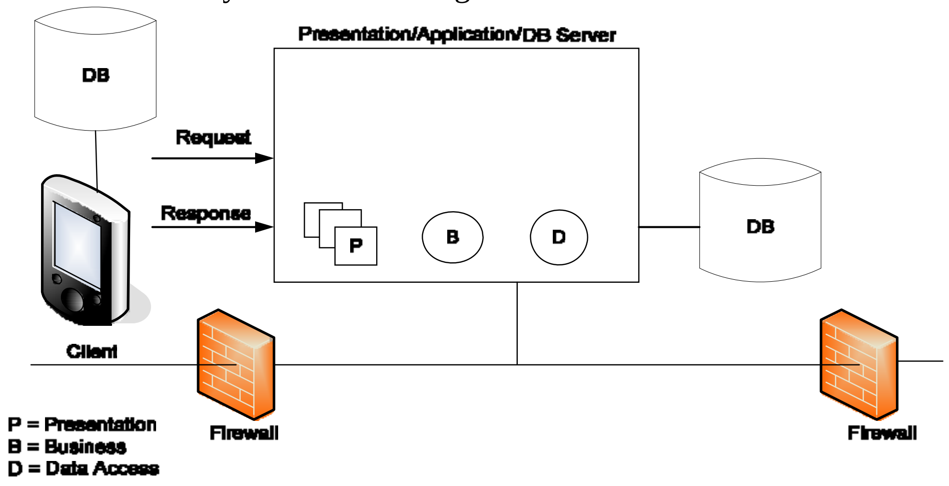


Fig. 2.6: One-Tier Architecture

Pros	**Cons**
i Very convenient	i Less scalable
ii Faster to develop and deploy	ii Hard to secure

3.2.2.3 Two-Tier Architecture

The three code layers exist on two (2) servers (Presentation/Application) and DB servers.

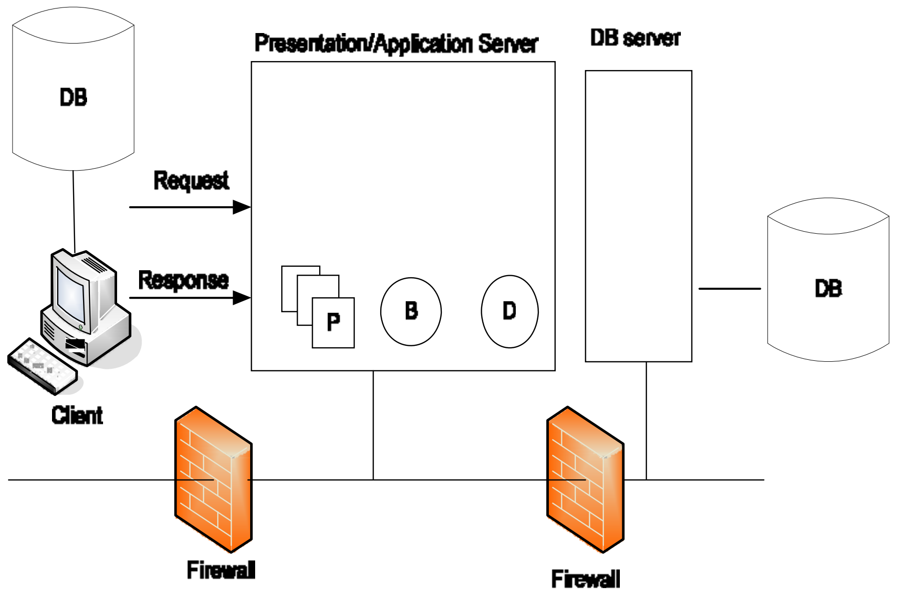


Fig. 2.7: Two-Tier Architecture

Pros	**Cons**
i Convenient	i Less scalable
ii Allow database server specialisation	ii Hard to secure
	iii More expensive

3.2.2.4 Three-Tier Architecture

The three code layers exist on three (3) servers (Presentation, Application and Database server.

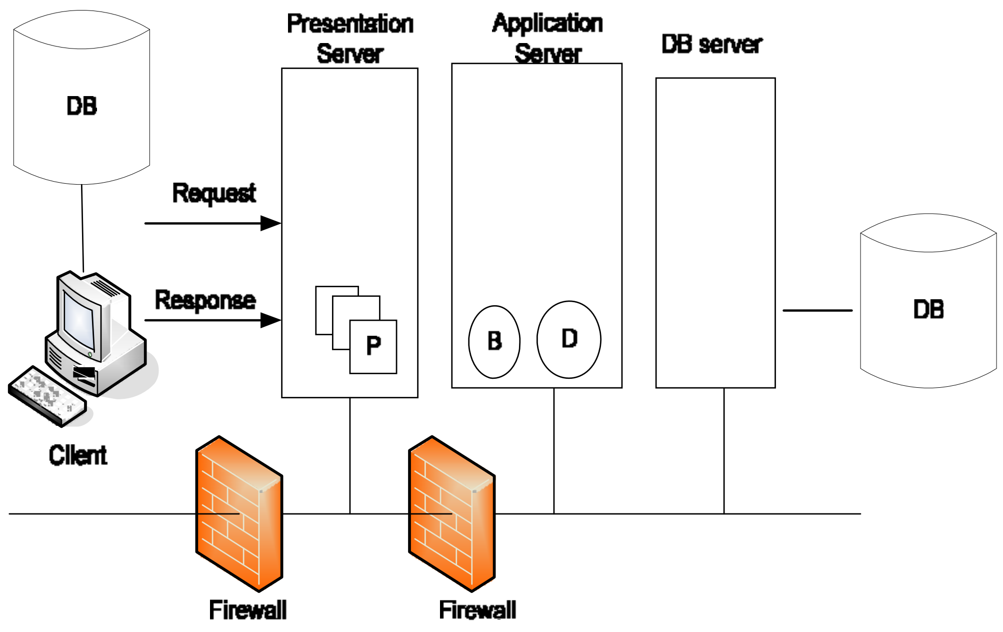


Fig. 2.8: Three-Tier Architecture

Pros	**Cons**
i Scalable	i Overkill
ii Secured behind firewalls and zones	ii More difficult to develop
iii Allows database server specialisation	iii More expensive

SELF-ASSESSMENT EXERCISE

i. Describe the tier architecture.
ii. Differentiate between tier and layer.

4.0 CONCLUSION

The World Wide Web is a global collection of interconnected documents on the Internet. The web itself is not the internet but part of the Internet that uses the Hypertext Transfer Protocol (HTTP) to display hypertext and images in a graphical environment. The World Wide Web is a system of internet servers that use **HTTP** (Hypertext Transfer Protocol) to transfer documents formatted in **HTML** (Hypertext Mark-up Language).

5.0 SUMMARY

In this unit, we have learnt:

- the structure of web application
- some web servers available
- the concepts of 3-tier and N-tier architecture
- the concepts of layers and tiers architecture.

6.0 TUTOR-MARKED ASSIGNMENT

i. Explain in detailed the structure of web application.
ii. Describe two most popular web servers.
iii. Distinguish between Layer and Tier.
iv. Explain the advantages and disadvantages of three tier architecture.

7.0 REFERENCES/FURTHER READING

Alberto, L. & Indra, W. (2000). Communications and Networking: Fundamental Concepts and Key Architecture. McGraw-Hill.

Richard, A. M.(2003). Introduction to Networking. McGraw-Hill.

William, B. (2000). Distributed Systems Networks .McGraw-Hill.

MODULE 2 WEB DESIGN

UNIT 1 INTRODUCTION TO WEB DESIGN

CONTENTS

1.0 INTRODUCTION

Web applications can be structured into 3-tier or n-tier architecture. The first tier is called the front-end, and it involves writing scripts in markup language. Examples of scripting language include Common Gateway Interface (CGI), Java Script, and Visual Basic Script (VB Script).

2.0 OBJECTIVES

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

- explain Markup language for client side programming
- describe Front-end scripting tools
- state HTML container and empty tags
- identify various formatting tags
- design a simple static web page.

3.0 MAIN CONTENT

Web Pages
A Web page is a file with an .htm or .html extension. It contains HTML, but might also contain other code stored on a server.

- Basically, web pages are of two types: static or dynamic
  Static Web pages contains only HTML tags. They only display information and there content does not change when requested by the user.
- Dynamic Web pages content changes depending on the user's request and preferences. Cannot be created using only mark-up language. It requires using Client-side code and Server-side code

3.1 Common Gateway Interface (CGI)

The Common Gateway Interface, or CGI, is a standard for communication between Web documents and CGI scripts you write. CGI scripting, or programming, is the act of creating a program that conforms to this standard of communication. A CGI script is simply a program that in some way communicates with your Web documents. CGI itself is not a language but its script can be written in any programming language. We shall discuss in details other scripting languages.

3.2 Mark-up Language

Standard Generalised Mark-up Language (SGML) is an International Standard Organisation (ISO) Standard meta language in which one can define mark-up languages for documents. It provides an abstract syntax that can be implemented in much different concrete syntax. For instance, angle bracket is used as tags delimiters. SGML is a system for defining markup languages. Authors *mark- up* their documents by representing structural, presentational, and semantic information alongside content. HTML is one example of a mark-up language**.**

Presently, standard markup language used by most web developer is Hypertext Markup Language (HTML).

3.3 Hypertext Mark-up Language (HTML)

This is a standard language for creating web pages. It provides tags that make documents look attractive using graphics, fonts and colours to enhance presentation. The basic building block of an HTML page is text, which are created using a Text Editor, while a Web Browser is used to test the application. The available text editors include Notepad or WordPad for Windows-based systems, while for Mac users, SimpleText is the HTML editor. UNIX users can use VI or Emacs.

Coupled with HTML, the needed tools for web design can be described as follows:

3.3.1 Hypertext Transfer Protocol (HTTP)

This is a protocol for communication on the web and it supports the client-server model. The client-side communicates with the server-side through HTTP.

3.3.2 Web Clients/browsers

The browsers interpret HTML documents. They are used to view texts, videos and audios as well as graphics files on web pages.

3.3.3 Hyperlinks

Hypertext links provide links to other portions of the web documents. When invoked, they forward requests to the server and the server responds with the appropriate response (document) to the client and it is displayed on the user's screen.

3.3.4 Uniform Resource Locator (URL)

URL is an address/file specifier. It locates the documents requested through hyperlinks. Similarly, it may be used to point to a query, image or a command. The **URL** (Uniform Resource Locator) is the global address of documents and other resources on the web e.g. http://www.who.int. The first part of the address indicates which protocol to use e.g. http. The second part of the address identifies the domain name or the internet address where the information is located.

Format:

- Scheme://host-domain/path/dataname
- http://www.openuniversity.ng/admissions/index.html

3.3.5 File Extensions Used in Web Design

File extension	**Data Type**
Html/htm	html text
text/txt	ASCII text
gif	composed graphics
jpeg/jpg	composed graphics
mpeg/mpg	digital video
avi	digital video
wav/au	digital video

3.4 Elements of HTML Structure

Basically, an HTML document is composed of tags or elements. The tags are classified into container and empty tags.

3.4.1 Container Tags

Tags that are composed of both ON **(<TAG>)** and OFF **(</TAG>)** tags are called container tags. They have both the beginning and the end tags. These tags wrap around and format texts in the document and hold or contain the texts between the two tags. The HTML, HEAD, TITLE and BODY tags are all container tags.

a <HTML> ……………… </HTML>
This is a prologue document identifier that is used at the beginning of an HTML document.

b <HEAD>……………….. </HEAD>
This element contains unordered collection of information about HTML document. This tag is optional as it is used to identify the portion of the file that describes the head element of the data.

c <TITLE>……………….. </ TITLE >
These tags are used to specify the title of the document on the title bar.
<HTML>
<HEAD>
<TITLE> Introduction to HTML </TITLE>
</HEAD>
</HTML>

d </BODY> ……………… </BODY>
The tags contain all the texts and images that are contained on the web page.
<html>
</body>My first experience with html coding </body>
</html>

e <P> ……………………………*</P>*
This is a paragraph tag that is used to demarcate one paragraph from another. Without the tags, the texts within the web document appear as on continuous piece, not segmented into paragraphs.

3.4.2 Empty Tags

All other HTML tags fall into this category called empty tags. These tags have only an ON tag, there are no OFF tags. The empty tags do not act on blocks of text, instead, they perform an action on their own. E.g. Horizontal rule <HR>, Line Break <BR>, etc.

<HR>	Horizontal Rule
<BR>	Line Breaks or Return
<!- Text -->	Comment

SELF-ASSESSMENT EXECISE

i. Mention four text editors and their providers.
ii. Mention five file extensions and their use in web application.
iii. With relevant examples describe container tags and empty tags.

3.5 Testing a Web Application

The steps involved in testing html codes are:

1. Open the Notepad (text editor)
2. Start/All Programs/Accessory/Notepad
3. Type the html codes
4. Save the code with the extension (htm or html)
5. The file name takes on an icon similar to the Internet explorer

6. Double-Click on the file icon to run the application.

3.6 Formatting Character Element

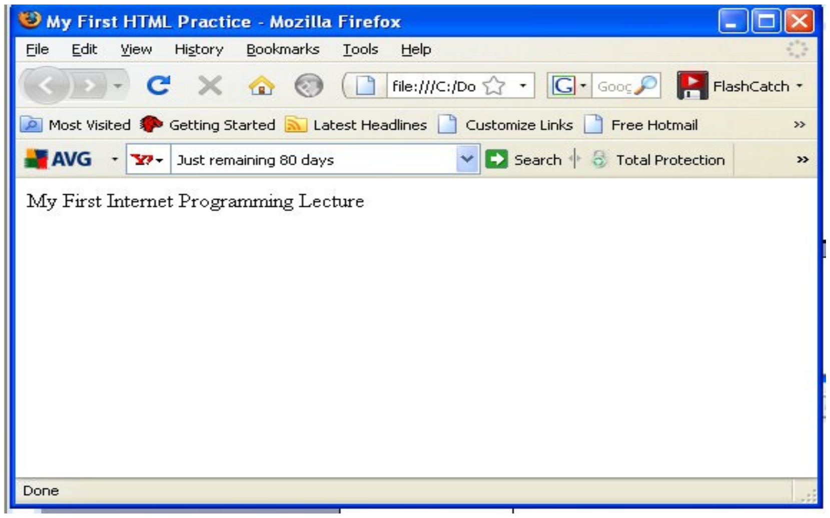
Generally, a web page is composed of a title bar, a menu bar and the content of the page. The texts contained within the title tags are displayed on the title bar, while the texts within the body tags are displayed as the contents of the web page. Thus, the following tags are used principally to format texts contained in web pages.

a	<B> text </B>	- Format text Bold
b	<I> text </1>	- Format text Italics
c	<STRIKE> text </STRIKE>	- Strike through text
d	<SUB> text or (no) </SUB>	- Format text Subscripts
e.	<U> text </U>	- Format text Underline
f.	<!This is CSC 312: Internet Concept and Web Design --->	- Comment

Example 1

- Now, let try an example on your Notepad. Type the following HTML code:
- <html>
- <Head>
- <Title>My First HTML Practice</title>
- </head>
- <Body>My First Internet Programming Lecture</Body>
- </Html>

Note: Don’t forget to save your file with .html


My First HTML Practice - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Do
FlashCatch
Most Visited Getting Started Latest Headlines Customize Links Free Hotmail
AVG Just remaining 80 days Search Total Protection
Done

My First Internet Programming Lecture

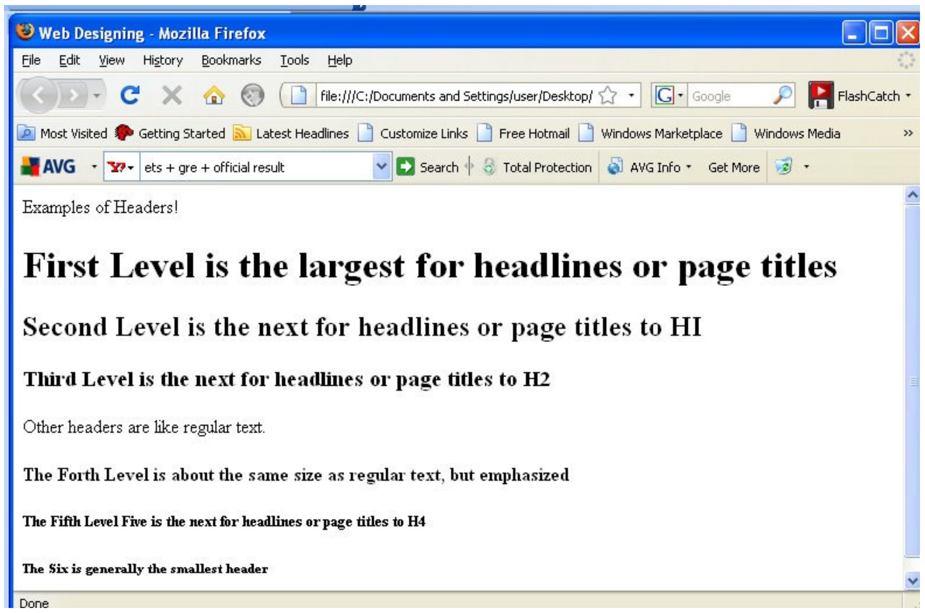
Fig.1.1: Output of Example 1

3.7 Creating Headers and Headlines

Header tags are containers tags. They double as paragraph tags ranging from level 1 to level 6. Headers enable designers to create different levels of emphasised headlines to organise documents. The following is an example:

Example 2

- <HTML>
- <HEAD>
- <TITLE> Web Designing</TITLE>
- </HEAD>
- <BODY>Examples of Headers!
- <Hl>First level is the largest for headlines or page titles</Hl>
- <H2> Second level is the next for headlines or page titles to HI </H2>
- <H3> Third level is the next for headlines or page titles to H2 </H3>
- <P>other headers are like regular text. </P>
- <H4>The Forth level is about the same size as regular text, but emphasised</H4>
- <H5>The Fifth level Five is the next for headlines or page titles to H4 </H5>
- <H6> The Six is generally the smallest header</H6>
- </BODY>
- </HTML>


Web Designing - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents and Settings/user/Desktop/
Google
FlashCatch
Most Visited Getting Started Latest Headlines Customize Links Free Hotmail Windows Marketplace Windows Media
AVG ets + gre + official result Search Total Protection AVG Info Get More
Done

Examples of Headers!

First Level is the largest for headlines or page titles

Second Level is the next for headlines or page titles to HI

Third Level is the next for headlines or page titles to H2

Other headers are like regular text.

The Forth Level is about the same size as regular text, but emphasized

The Fifth Level Five is the next for headlines or page titles to H4

The Six is generally the smallest header

Fig.1.2: Output of Example 2

3.8 Implicit and Explicit Text Emphasis

3.8.1 Explicit Styles

Explicit tags are referred to as physical tags. They specifically dictate how the text should physically appear to the browser. The basic explicit tags are containers that allow the user mark text as bold, italic, or underlined.

3.8.2 Implicit HTML Tags

Implicit styles are referred to as logical styles. They give browsers some level of freedom on how to display the text. These tags, like the header tags, are generally relative to one another, depending on the browser being used to view them.

Table 1: Basic Logical HTML Tags

Tag	Meaning	Remarks
<EM>, </EM>	Emphasis	Italic text
<STRONG>, < /STRONG >	Strong emphasis	Bold text
< TT>, </TT>	Teletype	Monospaced text

3.8.3 Preformatted Text

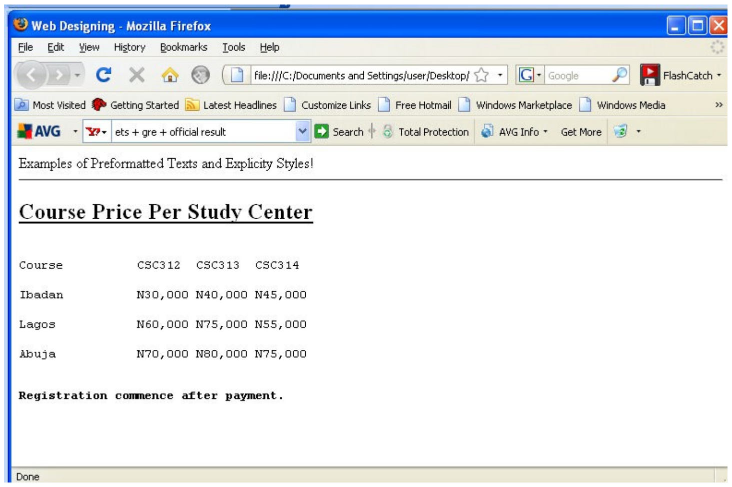
The <PRE> (preformatted text) tag is designed to allow you to keep the exact spacing and retains the format as presented, for instance, in a mathematical formula, or table creation

Example3

- <HTML>
- <HEAD>
- <TITLE>Web Designing</TITLE>
- </HEAD>
- <BODY>Examples of Preformatted Texts and Explicity Styles!
- <HR>
- <H2><U>Course Price Per Study Center</U></H2>
- <PRE>

Course	CSC312	CSC313	CSC314
Ibadan	N30,000	N40,000	N45,000
Lagos	N60,000	N75,000	N55,000
Abuja	N70,000	N80,000	N75,000

- <PRE>
- <B>Registration commence after payment. </B>
- </BODY>
- </HTML>



Web Designing - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents and Settings/user/Desktop/
Done

Examples of Preformatted Texts and Explicity Styles!

Course Price Per Study Center

```
Course          CSC312  CSC313  CSC314

Ibadan          N30,000 N40,000 N45,000

Lagos           N60,000 N75,000 N55,000

Abuja           N70,000 N80,000 N75,000
```

Registration commence after payment.

Fig.1.3: Example 3

3.9 HTML Lists

Lists are applied to groups of text and individual formatting tags within them. HTML lists are created using the form:

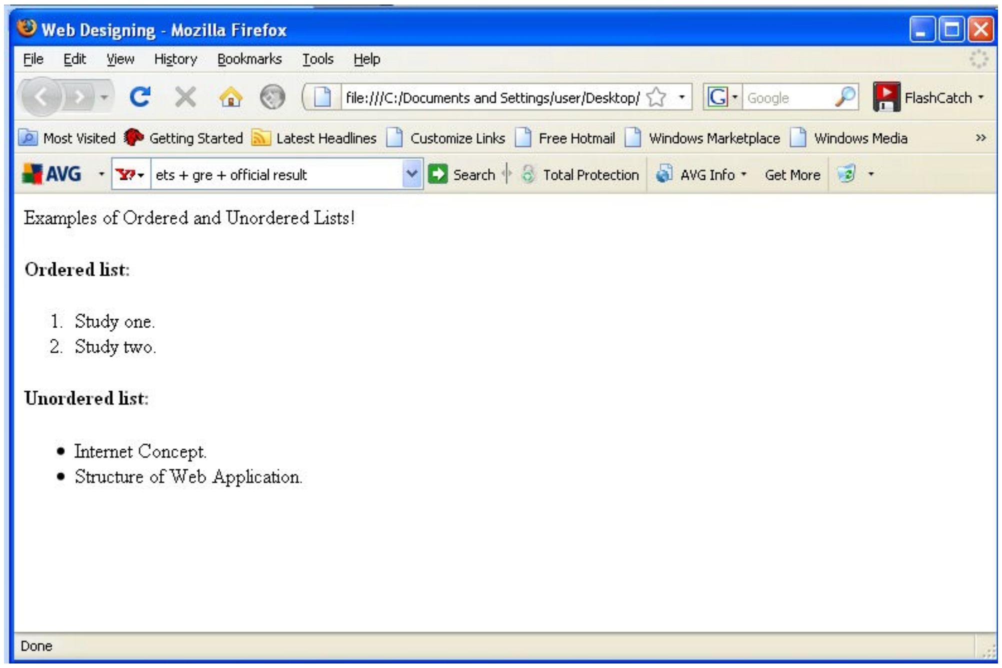
- <LIST TYPE>
- <ITEM> First item in list
- <ITEM> Second item in list
- <ITEM> Third item
- </LIST TYPE>

Where:

a. LIST TYPE = The types of list (ordered or unordered)
b. Ordered List = Numbered list, denoted by <OL>
c. Unordered List = Bulleted list, denoted by <UL>
d. ITEM = The individual items to be listed, denoted by <LI>

Example 4

- <HTML>
- <HEAD>
- <TITLE> Web Designing </TITLE>
- </HEAD>
- <BODY>Examples of Ordered and Unordered Lists!
- <H4>Ordered list: </H4>
- <OL>
- <LI> Study one.
- <LI> Study two.
- </OL>
- <H4>Unordered list: </H4>
- <UL>
- <LI> Internet Concept.
- <LI> Structure of Web Application.
- </UL>
- </BODY>
- </HTML>


Web Designing - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents and Settings/user/Desktop/
Google
FlashCatch
Most Visited Getting Started Latest Headlines Customize Links Free Hotmail Windows Marketplace Windows Media
AVG ets + gre + official result Search Total Protection AVG Info Get More
Done

Examples of Ordered and Unordered Lists!

Ordered list:

1. Study one.
2. Study two.

Unordered list:

- Internet Concept.
- Structure of Web Application.

Fig.1. 4: Example 4

3.9.1 Definition Lists (Indentations-Hanging)

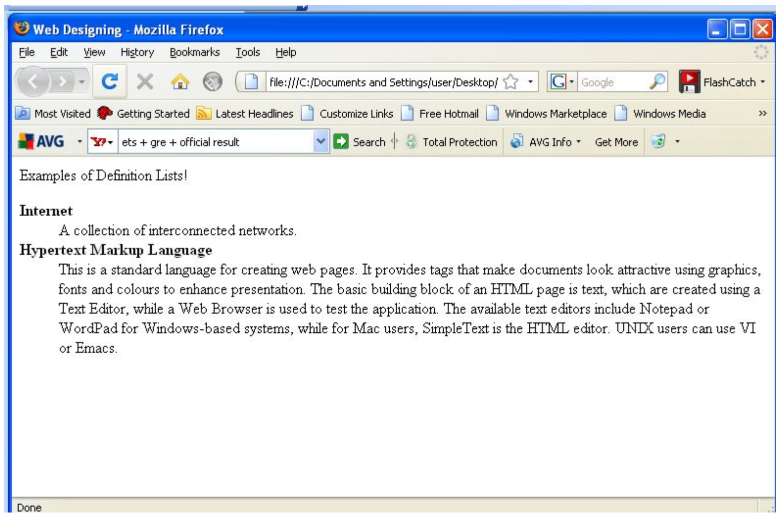
Definition lists are also known as "Indentation Hanging" They are purposely suited for definition of terms. They consist of a container tag <DL> (definition list) and two empty tags:

a. <DT> (definition term or the term to define); and
b. <DD> (definition or the definition provided).

The <DT> tag is designed (ideally) to fit on a single line of your web page, although it may wrap to the beginning of the next line if necessary. The <DD> tag will accept a full paragraph of text, continuously indented beneath the <DT> term.

Example 5

- <HTML>
- <HEAD>
- <TITLE> Web Designing</TITLE> </HEAD>
- <BODY>Examples of Definition Lists!
- <DL>
- <DT><B>Internet</B>
- <DD>A collection of interconnected networks.
- <DT><B>Hypertext Markup Language</B>
- <DD> This is a standard language for creating web pages. It provides tags that make documents look attractive using graphics, fonts and colours to enhance presentation. The basic building block of an HTML page is text, which are created using a Text Editor, while a Web Browser is used to test the application. The available text editors include Notepad or WordPad for Windows-based systems, while for Mac users, SimpleText is the HTML editor. UNIX users can use VI or Emacs.
- </DL>
- </BODY>
- </HTML>


Web Designing - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents and Settings/user/Desktop/
Done

Examples of Definition Lists!

Internet
A collection of interconnected networks.
Hypertext Markup Language
This is a standard language for creating web pages. It provides tags that make documents look attractive using graphics, fonts and colours to enhance presentation. The basic building block of an HTML page is text, which are created using a Text Editor, while a Web Browser is used to test the application. The available text editors include Notepad or WordPad for Windows-based systems, while for Mac users, SimpleText is the HTML editor. UNIX users can use VI or Emacs.

Fig.1. 5: Example 5

SELF-ASSESSMENT EXERCISE

i. Describe the use of four HTML formatting elements.
ii. In a simple web page, demonstrate the use of HEAD tags and DEFINITION tags.

4.0 CONCLUSION

Web applications can be structured into 3-tier or n-tier architecture. The first tier is called the front-end which involves writing scripts in mark-up language.

5.0 SUMMARY

In this unit, we have learnt:

- understanding of the markup language for client side programming
- understanding of the front-end scripting tools
- structure of HTML document
- understanding of HTML container and empty tags
- various formatting tags
- ordered, unordered and definition lists
- creation of hyperlinks and insert graphics on web pages
- creation of forms and tables in web pages
- understanding of how to design a simple static web page.

6.0 TUTOR-MARKED ASSIGNMENT

i. Enumerate the steps involved in testing web application.
ii. State the HTML tags that can be used to describe the table below

		Individuals		Organisations	
S/N	Features	Yes	No	Yes	No
1	Web Presence	148 (47%)	196 (50.3%)	88 (88%)	11 (11%)
2	Internet Access	283 (72.6%)	100 (25.3%)	97 (97%)	2 (2%)
3	Payment Cards Availability	141 (36.2%)	241 (61.8%)	NA	NA
4	Participation in online purchase	82 (21%)	296 (75.9%)	***	***

iii. Design a simple web page to list out five courses in your program.

7.0 REFERENCES/FURTHER READING

Michael, M. (2001). " HTML and XML for beginners"

http://dev.opera.com/articles/view/4-the-web-standards-model-html-css-a/

http://www.corewebprogramming.com

UNIT 2 ADDING GRAPHICS TO WEB PAGES

CONTENTS

1.0 INTRODUCTION

The previous unit introduced you to HTML scripts. This unit discusses how you can add graphics to web pages. This unit also discusses how you can create hypertext and hypermedia link, table in HTML and also the HTML form.

2.0 OBJECTIVES

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

- explain how to work with Web graphics
- produce hypertext and hypermedia links
- prepare HTML Form
- produce pop-up and scrolling menus
- prepare tables in HTML documents.

3.0 MAIN CONTENT

Types of Web Graphics File

There are two types graphics file: CompuServe Graphics Interchange Format (GIF) and Joint Photographic Experts Group (JPEG).

GIFs are compressed graphics that tend to lose less image clarity than JPEGs. GIFs are particularly well suited for images that have smaller colour palettes (256 colours or fewer).

JPEG format is widely being used by web designers as it can be viewed in most new graphical browsers without the help any special application. JPEG has the advantage of viewing images that have more colours (up to 16.7 million).

JPEGs are a little flexible than GIFs, that is high rate of compression results in slightly lower image quality. JPEG files are usually smaller and easily transmittable over the Internet more quickly.

3.1 Working with Web graphics

Web graphics can be obtained in different ways:

a. Creating graphics in a graphics application and saved as GIFs or JPEG.
b. Downloading public-domain graphics such as the Internet or public-domain clipart collections (that are available on CD-ROM).
c. Using scanned photographs. Existing photographs can be scanned into digital form for use on the site.
d. Using digital cameras to snap pictures that can be downloaded directly from the camera to your computer.

3.2 Embedding Graphics in Web Pages

The image tag <IMG> is used to insert images into the web document. This is done in two ways depending on the location of the image file:

3.2.1 Relative Path URL

This is used if the image file is located in the same folder as the HTML document.

Format: <IMG SRC="image URL">
E.g. <IMG SRC="openImage.gif">

3.2.2 Absolute Path URL

This is used if the image file is located on a remote site outside the folder that contains the HTML documents.

Format: <IMG SRC="path/filename"> Eg. <IMG
SRC="http://www.openuniversity.com/lectures/openImage.gif">
Or <IMG SRC="c:\My Document\lectures\openImage.gif">

3.3 Other Image Tag <IMG> Attributes

1 **The ALT Attribute**
The ALT attribute for the <IMG> tag gives the description of the graphics in case the browser could not display the image. It informs the users that the graphic exists and explains what the graphic is all about.
E.g. <IMG SRC="openImage.gif" ALT= “National Open University Logo”>

2 **The ALIGN Attribute**
It specifies the alignment of other elements such as top, middle, or bottom of the graphic.
Eg. <IMG SRC="image URL" ALIGN="direction">

Example 6

- <html>
- <head>
- <title> Web Design</title>
- </head>
- <body>
- <br>
- <h3>Internet Programming</h3>
- <img border="0" src="CandPix.JPG" width="75" height="50" align=”left”>
- </body>
- </html>

3.4 Creating Hypertext and Hypermedia Links

Links are created in HTML using the anchor **<A>** tag
The Anchor <A> tag is used for creating hypertext and hypermedia links. The links are embedded between <A> and </A>.

Format
<A HREF="URL"> Text describing link</A>

HREF is an attribute for the anchor <A> tag. The hyperlink usually appeared underlined and in different colour from the rest of the documents.

Let look at the following, to see how to create a link:

1. **Creating a link to the documents on the local computer**
   E.g. The Next and Back buttons.
   Format: <A HREF= "products.html"> Product Information</A>
2. **Creating a link to other documents on the Internet**
   In this case, the URL must be specified.
   <A HREF="http://www.openuniversity.com/admission.html"> Admission Information</ A>
3. **Creating a link to other parts of the same document (Section link)**
   This is useful for creating a link to other sections of the web page such as top, bottom, etc without scrolling from top to bottom.
   Format: <A HREF="#section name">Link to another section of the document</A>
4. **Adding the <BASE> Tag**
   The tag is used when web pages are stored in different sub-folders within a main folder. Thus, the tag directs the browser to search the main folder on the machine as against the current directory.

   Format: <BASE HREF="absolute URL">
   The <BASE> tag is designed to appear only between the <HEAD> tags

```
<HEAD>
<BASE HREF= "http://openuniversity.com/">
< TITLE> Study One</TITLE>
</HEAD>
<BODY>
<A HREF="practice.html">Back to Index</A>
 </BODY>
```

5 Hyperlinks for E-Mail Messages

A hyperlinked e-mail address is accomplished using the mailto: type of URL.
Format: <A HREF="mailto:emailaddress">text</A>
e.g. <AHREF =mailto:oye@yahoo.com>Send me e-mail</A>

Example 7: You are to create a simple web application to display an image, navigate pages back and forward. Also, create an hyperlink to e-mail.

- <html>
- <head>
- <title> Web Programming</title>
- </head>
- <body>
- <br>
- <h3>Navigating through Web Pages</h3>
- <img border="0" src="Picture 0009.JPG" width="75" height="50" >
- <BR>
- <HR>
- <A HREF= "Example4.html"> Back to application 4</A> <BR>
- <A HREF="mailto: oyelade@yahoo.com"> Send Mail to Author</A> <BR>
- </body>
- </html>

3.5 Creating HTML Form

A form is an area that can contain form elements. Forms are used on the web to accept information from the users. Doing any registration online, you have to fill a form which its information is passed into the server. The server hands it to a script that is designed to process the data and generate an HTML page in response or store the information in the database.

Forms are created within an HTML document using the <FORM> container tags. The <FORM> tag has to attributes: and ACTION.

3.5.1 Form Elements

Form elements are elements that allow the user to enter information (like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.

Format: <FORM METHOD="*how to send*" ACTION="*URL of script*">...form data... </FORM>

Where: - METHOD value "*how_to_send*" = Post or Get

"*URL of script*" = The address of the middleware to process the information

The METHOD attributes refer to how the input data is transferred, while the ACTION attribute refers to the relative address of the script to process it.

The values for METHOD attributes are POST and GET, the difference between them are as explained below:

a. The POST method will send data stream to the script when requested.
b. The data sent by the POST method is invisible.

While
a. The GET method will send data in form of URL.
b. The data sent by the GET method is displayed on the address bar.
c. Data can be sent by the GET method without a form, but by editing the URL.

Example
$variable=$ POST[‘variable’];
or
$variable=$ GET[‘variable’];

Where ‘variable’= the field(s) in the HTML form
And $variable= the field(s) in the database table with the dollar sign

3.5.2 Text Fields and Attributes of a Form

1 Text Area

This is used to accept multiple lines of text from a user. It specifies the number of rows and columns.

Format:

- <TEXTAREA NAME="*variable_name*" ROWS="*number*" COLS="*number*"> default text </TEXTAREA>

Where

- NAME = The variable name assigned to the TEXTAREA. That is, the name passed to the database.
- ROWS = The number of rows required for the text box.
- COLS = The number of columns required for the text box.
- Default Text = The text to display in the text box as default.

Example

- <TEXTAREA NAME="shortnote" ROWS="5" COLS="60">
  Type your lecture here.
  </TEXTAREA>

2 The <INPUT> Tag

This is used to accept the user’s input from the form. It is the most commonly used form attribute.

Format

<INPUT TYPE="type of box" NAME="variable" SIZE="number" MAXLENGTH="number" VALUE = “Your Entry”>

Where

- TYPE = denote the types of input (Text, Password, Checkbox, Radio, Reset, and Submit)
- NAME = The name assigned to the variable
- SIZE = The field length of the entry
- MAXLENGTH = The maximum length of the entry permitted
- VALUE = The default value to be displayed in the form.

Example

Last Name: <INPUT TYPE="TEXT" NAME="first_name" SIZE="15" MAXLENGTH="25”>

Password

This is identical to the TEXT option, but the entries are replaced with bullet points or a similar scheme (chosen by the browser) to prevent the words from being displayed in plain text.

Format

Enter Password: <INPUT TYPE="PASSWORD" NAME="password" SIZE="8" MAXLENGTH="8">

3 **Checkbox**

Checkboxes are well-suited for data entry that involves two possible values for a given choice. A default choice is checked by using the attribute CHECKED.

Example

- Sex: <INPUT TYPE="CHECKBOX" NAME="sex" CHECKED> Male
- <INPUT TYPE="CHECKBOX" NAME="sex"> Female<BR>

The checked item is the default and more than one item could be checked since the CHECKBOX evaluates each item separately from any other ones.

4 **Radio**

The **Radio** button is designed to accept only one response from among several options, unlike Checkbox that can have multiple choices. RADIO attributes are similar to the CHECKBOX attributes.

Example

- Faculty:<INPUT TYPE="RADIO" NAME="faculty" VALUE="Art”> Art
- <INPUT TYPE="RADIO" NAME=" faculty" VALUE="Science"> Science
- <INPUT TYPE="RADIO" NAME=" faculty" VALUE="Education"> Education <BR>

5 **Reset**

This tag is used to clear all entries of an HTML form. It creates a push button (named with the VALUE string) that resets all of the elements in that particular FORM to their default values

Example

- <INPUT TYPE = "RESET" NAME="RES" VALUE="RESET">

6 **Submit**

This is similar to the RESET button but differs in usage. It is used to transfer or store the data entered into the HTML form to the database. The VALUE attribute is used to rename the button (the name that appears on the button).

Example

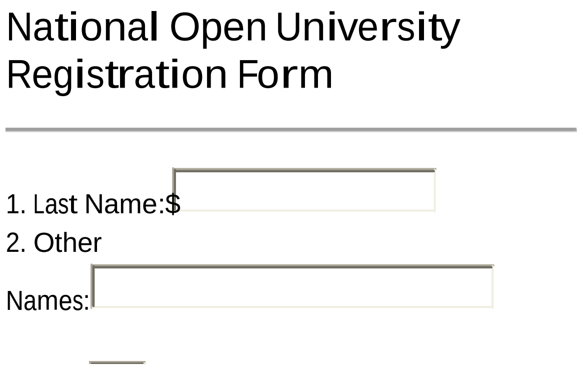
- <INPUT TYPE="SUBMIT" NAME="SUB" VALUE="SUBMIT" <BR>

Example 8: In this example, you are required to design a web application that accept: i) Name, ii) Age, iii) Sex, iv) Programme, v) Level, and vi) Study centre as inputs from the user, save the inputs and also clear (reset) the form. Add other esthetics to beautify your form.

Solution: open your Notepad and type the following lines of code:

- <HTML> <!..........Student Registration....>
- <TITLE> Registration Form </TITLE>
- <BODY>
- <H1> National Open University Registration Form</H1>
- <FORM METHOD ="POST"><HR>
  1. Last Name:$<INPUT TYPE ="TEXT" SIZE="25" NAME="L_Name"><BR>
  2. Other Names:<INPUT TYPE ="TEXT" SIZE="40" NAME="O_Name" ><BR><BR>
  3. Age: INPUT TYPE ="TEXT" SIZE="3" NAME="O_Name" ><BR><BR>
  4. Sex: <INPUT TYPE="CHECKBOX" NAME="sex" CHECKED> Male
     <INPUT TYPE="CHECKBOX" NAME="sex"> Female<BR><BR>
  5. Program:<INPUT TYPE="RADIO" NAME="Program" VALUE="Accounting" CHECKED> Accounting
     <INPUT TYPE="RADIO" NAME="Program" VALUE="Business Admin"> Business Administration
     <INPUT TYPE="RADIO" NAME="Program" VALUE="Chem Eng"> Chemical Engineering<BR>
- <INPUT TYPE="SUBMIT" NAME="SUB" VALUE="SUBMIT" <BR>
- <INPUT TYPE = "RESET" NAME="RES" VALUE="RESET">
- </FORM>
- </BODY>
- </HTML>

The output should like this below:


National Open University Registration Form

1. Last Name:$
2. Other Names:

3.6 Creating Pop-Up and Scrolling Menus

3.6.1 Listbox

The <SELECT> tag is a container tag that allows a user to select from a Listbox .

Format:

<SELECT NAME="*variable*"> <OPTION SELECTED VALUE="*value*"> Menu text <OPTION VALUE="*value*"> Menu text ... </SELECT>

Where

NAME = The name of the variable
SELECTED VALUE = The default value
OPTION VALUE = Other possible option
Etc

SELF-ASSESSMENT EXERCISE

i. Add the following lines of code just before the “SUBMIT” button in to your program in **Example 8** and produce the required output.
   a. Level:<SELECT NAME="LEVELS">
   b. <Option Selected Value = 100> 100
   c. <Option Value = 200> 200
   d. <Option Value = 300> 300
   e. <Option Value = 400> 400
   f. </SELECT><BR>
   g. Address:<HR><TEXTAREA NAME="Addr" ROWS="4" COLS="40">
   h. Enter Your Home Address.</TEXTAREA><BR><BR>

NOTE: To allow multiple selections the world MULTIPLE is added to the first line as:

<SELECT NAME="*variable*" MULTIPLE>

3.7 Creating Tables in HTML Documents

Just as you can create table with rows and columns in various text applications, you can also create tables in HTML using the tags and its attributes.

The table tags include:

<TABLE>	HTML Table	</TABLE>
<TR>	Table Row	</TR>
<TD>	Table Data	</TD>
<TH>	Cell Header	</TR>
<CAPTION>	Caption of the Table	</CAPTION>

Format

- <TABLE>
  <CAPTION>Caption text for table</CAPTION>
  <TR>
- <TH>column1</TH>
- <TH>column2</TH>
- <TH>column3</TH>
- </TR>
  <TR>
- <TD>row1data1</TD>
- <TD>row1data2</TD>
- <TD>row1data3</TD>
- </TR>
  <TR>
- <TD>row2data1</TD>
- <TD>row2data2</TD>
- <TD>row2data3</TD>
- </TR>
  ...
  </TABLE>

3.7.1 The <TABLE> Tags' Attributes

1 ALIGN: The ALIGN attribute specifies the location of the table in the browser window. Valid options are ALIGN=LEFT and ALIGN=RIGHT.

2 WIDTH: The WIDTH attribute specifies the width of the table in the browser window.

3 COLS: The COLS attribute specifies the number of columns in your table, allowing the browser to draw the table as it downloads.

4 BORDER: The BORDER attribute defines the width of the border surrounding the table.

5 CELLSPACING: The CELLSPACING attribute specifies the space to include between the walls of the table and between individual cells.

6 CELLPADDING. The CELLPADDING attribute specifies how much space to give data elements away from the walls of the cell.

7 <CAPTION> Tags' Attributes

The <CAPTION> tag has one attribute, ALIGN. It is either ALIGN="TOP" or ALIGN="BOTTOM". The TOP is the default setting.

We shall see examples of all these later.

3.7.2 Table Data and Rows Attributes

There are four attributes ALIGN, VALIGN, COLSPAN, and ROWSPAN.

Format:

- <TABLE BORDER= # of pixels> </TABLE> <!.... 0 to 5…>
- <CAPTION ALIGN= "alignment"</CAPTION <!....TOP, BOTTOM….>
- <TD ROWSPAN= "#"</TD> <! . A cell that spans # rows..>
- <TD COLSPAN="#"</TD> <! . A cell that spans # cols..>
- <TH ROWSPAN="#"</TH> <! A cell HEADER that spans # rows..>
- <TH COLSPAN="#"</TH> <!. A cell HEADER that spans # rows..>
- <TD ALIGN="alignment"</TD> <!. Aligns Left, Right, Centre..>
- <TD VALIGN="alignment"</TD> <!. Aligns Top, Middle, Bottom..>
- <TD WIDTH="#" HEIGHT= "#" </TD><!. Sets width and height of cells..>
- <TH WIDTH="#" HEIGHT= "#" </TD><!. Sets width and height of header..>

Note: You can also embed Links and Graphics within a table

Example 9

- <HTML> <!..........Student Registration....>
- <TITLE> Using Tables and Its Attributes </TITLE>

- <BODY>
- <TABLE BORDER>
- <TR>
- <TH>Student</TH>
- <TH>CSC 311</TH>
- <TH>CSC 312</TH>
- <TH>GPA</TH>
- </TR>
- <TR>
- <TH>Adeoye T. Y.</TH><TD>80</TD>
- <TD>75</TD>
- <TD ROWSPAN="3">N/A</TD>
- </TR>
- <TR>
- <TH>Susan N.</TH><TD>70</TD>
- <TD>90</TD>
- </TR>
- <TR>
- <TH>Babs L. </TH><TD COLSPAN="2">Dropped Course</TD>
- </TR>
- </TABLE>
- </HTML>

The output looks like this below:

Student	**CSC 311**	**CSC 312**	**GPA**
Adeoye T. Y.	80	75	N/A
Susan N.	70	90	
Babs L.	Dropped Course		

SELF-ASSESSMENT EXERCISE

i. Distinguish between relative and absolute URI path.
ii. To Example 8 above, include the logo of National Open University and align it to the left.

4.0 CONCLUSION

This unit discusses how you can add graphics to web pages and how you can create hypertext and hypermedia link, table in HTML and also the HTML form.

5.0 SUMMARY

In this unit, we have learnt:

- how to work with Web graphics
- creating hypertext and hypermedia links
- creating HTML Form
- creating pop-up and scrolling menus and
- creating Tables in HTML Documents.

6.0 TUTOR-MARKED ASSIGNMENT

i. You have just been employed as a web developer in an Order Processing Organisation. Using HTML code, design a form to Register Customers for Order Placement.
ii. Include a link called HOME to 1 above.
iii. Create a web page titles home consisting of the Organisation logo and a brief description of the Company.

7.0 REFERENCES/FURTHER READING

Michael Morrison (2001), “ HTML and XML for beginners”

http://dev.opera.com/articles/view/4-the-web-standards-model-html-css-a/

http://www.corewebprogramming.com.

UNIT 3 CASCADING STYLE SHEET (CSS)

CONTENTS

1.0 INTRODUCTION

Cascading Style Sheets (CSS) is a style sheet language used to describe the presentation semantics (i.e. the look and formatting) of a document written in a markup language. CSS is designed primarily to enable the separation of document content (written in HTML or a similar markup language) from document presentation, including elements such as the colours, fonts, and layout. This helps to improve content accessibility and control in the separation of presentation characteristics. CSS also helps to improve flexibility and enable multiple pages to be shared thereby reducing complexity and repetition in the structural content. CSS can also allow the same markup page to be presented in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser or screen reader) and on Braille-based, tactile devices.

CSS specifies a priority scheme to determine which style rules apply if more than one rule matches against a particular element. In this so-called cascade, priorities or weights are calculated and assigned to rules, so that the results are predictable.

2.0 OBJECTIVES

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

- explain the structure of web application design style sheet rules
- create external and inline style specification
- format HTML with CSS font and text proprieties
- explain foreground and background property.

3.0 MAIN CONTENT

Brief History of CSS
Style sheets have existed in one form or another since the beginnings of SGML in the 1970s. CSS were developed as a means for creating a consistent approach to providing style information for web documents. Until this present moment, CSS has gone through three developmental stages each with its specified focus.

- CSS, Level 1 (1996): concerned with applying simple styles to HTML elements. http://www.w3.org/TR/REC-CSS1
- CSS, Level 2 (1998): supports media-specific styles sheets (visual browsers, aural device, printers, and Braille devices). http://www.w3.org/TR/REC-CSS2
- CSS, Level 3 (currently under development): focused on modularisation of the CSS specification. http://www.w3.org/TR/CSS3-roadmap/

3.1 CSS Syntax

Designing simple style sheet is easy but requires a little understanding of HTML. The CSS rule is made up of three parts: a selector, a property, and a value:

```
Format:     selector{ property:value }
Or
selector { property1: value1;
           property2: value2;
           ….
           property; valueN }
```

Where:

a. Selector = normal HTML element/tag you wish to define. It is the link between the HTML document and the style sheet.
b. Property = the attribute of the tag intended to change.

NOTE: each property must have and a value and the property and value are separated by colon (:)

Example:

```
 H1 {text-align: center;
        colour; black }
```

3.1.1 Grouping

You can group selector. Separate each selector with a comma. For example:

```
H1,H2,H3,H4
{
Colour:blue
}
```

3.1.2 The Class Selector

With the class selector you can define different styles for the same type of HTML element.

For instance, you may like to have two types of paragraphs in your document: one right aligned and one center-aligned paragraph. You will have to use the class attribute in your HTML document

```
<p class="right">This paragraph will be right-aligned.</p>
<p class="center">This paragraph will be center-aligned.</p>
```

More than one class can also be applied to a given element. For example:

```
<p class=”center bold”>this paragraph is aligned center and bolden</p>
```

The paragraph above will be styled by the class “center” AND the class “bold”.

3.2 Defining Global Style Class

You can define a style that will be used by all HTML elements having a certain class. This is done by omitting the tag name in the selector.

Format:

- .green { colour: green; font-weight: italics }. To use this, simple specify the class in CLASS attribute of the HTML.
- Example:
- <h3 class="green">This heading will be coloured green</h3> <p class="green">This paragraph will also be coloured green.</p>

Here, both h3 and p element have class =”center”. This means that both elements will follow the rules in the “.center” selector:

Note: Do not start a class name with a number

3.3 Adding Styles to Elements with Particular Attributes

Styles can also be applied to HTML elements with particular attributes For example, the style rule below will match all input elements that have a type attribute with a value of “text”

Input[type=”text”] {background-colour: blue}

3.3.1 The Id Selector

An ID is like a class but can be applied only once in a document. The id selector is defined as #.

Format for id style rule:

- #green { colour:green}

Note: Do not start an ID name with a number!

3.3.2 CSS Comments

As in mark-up languages and programming languages comments are used to explain your code. A CSS comment begins with “/*”, and ends with “*/”.

Example:
- /* This is a comment*/
- Body
- {background: blue}

Having gone through CSS rules (syntax) now let us see how to include CSS in our HTML code.

3.4 Inserting CSS

There are three ways to inserting a style sheet:

a. External style sheet
b. Internal style sheet
c. Inline style

3.4.1 External Style Sheet

Using an external style sheet, the CSS code is written in a file saved with CSS. An external style sheet is ideal when the style is applied to many pages. Changing the look of an entire web site is done by changing only the CSS file. Each page must link to the style sheet using <link> tag. The <link> is placed inside the head section.

Example:

- <head>
- <title>Cascading Style Sheet Class</title>
- <link rel=”stylesheet” type=”text/css” href=”myCSSFile.css” />
- </head>
- The <link> tag has three attribute: rel, type and href. It is an empty tag so “/” must precede the second angle bracket (“>”).

3.4.2 Internal Style Sheet

This should be used when the CSS rule is to be applied to a single Web page. You define the internal styles in the head section of an HTML page, by using the <style> tag.

Example:

```
<head>
<style                                              type="text/css">
hr                                                  {colour:sienna}
p                                                 {margin-left:20px}
</style>
</head>
```

3.4.3 Inline Styles

To use an inline styles you use the style attribute in the relevant tag. The style attribute can contain any CSS property.

Example:<p style=”colour:brown; margin-left:20px”>This is a paragraph</p>

Using an inline styles loses many of the advantages of style sheets by mixing content with presentation. This method is not recommended to be used often.

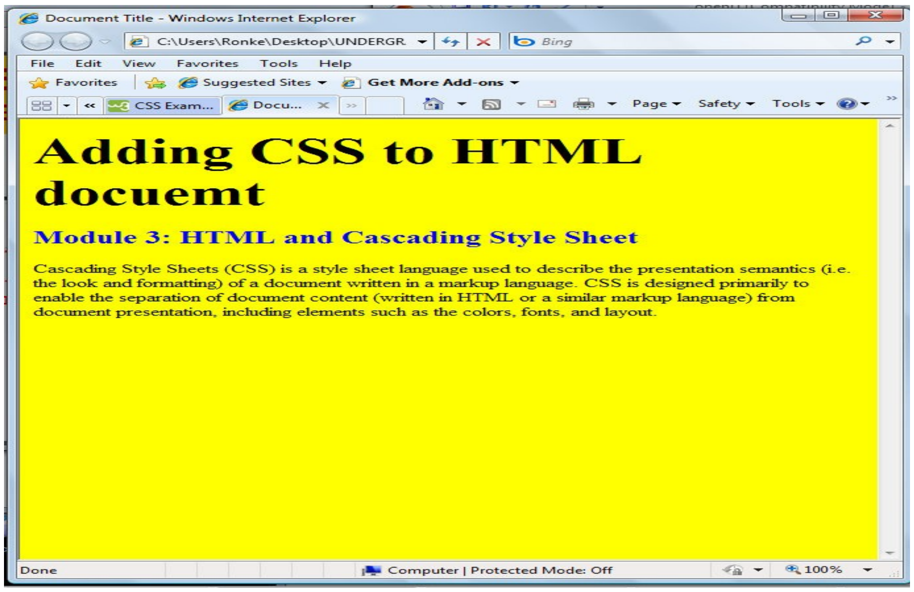
Example 1: Example of using internal style sheet

- <HEAD>
- <TITLE>Document Title</TITLE>

- <STYLE TYPE="text/css">
- <!--
- body {background-colour: yellow}
- h1 {font-size: 36pt}
- h2 {colour: blue}
- p {margin-left: 50px}
- -->
- </STYLE>
- </HEAD>
- <BODY>
- <H1>Adding CSS to HTML document</H1>
- <H2>Module 3: HTML and Cascading Style Sheet</H2>

Cascading Style Sheets (CSS) is a style sheet language used to describe the presentation semantics (i.e. the look and formatting) of a document written in a markup language. CSS is designed primarily to enable the separation of document content (written in HTML or a similar markup language) from document presentation, including elements such as the colours, fonts, and layout.

</BODY>
</HTML>


Document Title - Windows Internet Explorer
C:\Users\Ronke\Desktop\UNDERGR
Bing
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Get More Add-ons
CSS Exam...
Docu...
Page
Safety
Tools
Done
Computer | Protected Mode: Off
100%

Adding CSS to HTML docuemt

Module 3: HTML and Cascading Style Sheet

Cascading Style Sheets (CSS) is a style sheet language used to describe the presentation semantics (i.e. the look and formatting) of a document written in a markup language. CSS is designed primarily to enable the separation of document content (written in HTML or a similar markup language) from document presentation, including elements such as the colors, fonts, and layout.

Fig.3.1: Result of Example 1

Looking at the above illustration, the style rules:

- **Body {background-colour: yellow}**
- **h1 {font-size: 36pt}**
- **h2 {colour: blue}**

- **P {margin-left: 50px}** are placed inside style tags <STYLE> </STYLE> in the HEAD part of HTML document. The CSS rules are also written inside the HTML comment tag because the browser will not interpret any element that is not HTML tag.

SELF-ASSESSMENT EXERCISE

i. Carry out example1 using External Style Sheet method. Also, add background image to your work.
ii. Add a comment to include your name on the page and change the background colour to brown.

4.0 CONCLUSION

CSS is a style sheet language used to describe the presentation semantics of a document written in a markup language. It is designed to enable the separation of document content (written in HTML or a similar markup language) from document presentation, which helps to improve content accessibility and control in the separation of presentation characteristics. CSS also helps to improve flexibility and enable multiple pages to be shared.

5.0 SUMMARY

In this unit, we have learnt:

- specify style sheet rules
- create external and inline style specification
- format HTML with CSS font and text proprieties
- control foreground and background property.

6.0 TUTOR-MARKED ASSIGNMENT

i. Using Internal CSS format in your Tutor- Marked Assignment 1 in module two unit two to include background colour, heading of colour blue with 32pt.
ii. Marge the page to 100px left margin.

7.0 REFERENCES/FURTHER READING

Michael Morrison (2001), “ HTML and XML for beginners”

http://dev.opera.com/articles/view/4-the-web-standards-model-html-css-a/

http://www.corewebprogramming.com

UNIT 4 CASCADING STYLE SHEET PROPERTIES

CONTENTS

1.0 INTRODUCTION

The previous unit introduced us to the CSS rules. This unit discusses properties that CSS use to define the outlook of web pages. Such property include: text, font, box model, border, outline etc. These properties helps designer to control the look of their web pages. It also discusses the type of precedence that can be given to CSS rules.

2.0 OBJECTIVES

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

- outline the use of: Background, Text, Font, Box Model, font, Border Outline Margin, Padding, List, and Table property
- use more than one style sheet for a document (cascade)
- describe how to create Style Sheet Layer
- enumerate the Style Sheet Precedence rules.

3.0 MAIN CONTENT

When using CSS, more than one style sheet can influence the presentation of a web page simultaneously. There are two main reasons for this:

1 **Modularity**
A style sheet designer can combine several (partial) style sheets to reduce redundancy:
- @import url(http://www.openuniversity.org/css1);
- @import url(http://www.openuniversity.org/css11);
- H1 { colour: red } /* override imported sheets */

2 Author/Reader balance

Both readers and authors (script writer) can influence the presentation through style sheets. To do so, they use the same style sheet language thus reflecting a fundamental feature of the web.

- However, conflicts may sometimes arise between the style sheets that influence a web page presentation. To resolve this, conflict resolution is based on giving each style rule a priority. By default, the priority of the reader's rules is less than the priority of rules in the author's documents i.e., if there are conflicts between the style sheets of an incoming document and the reader's personal sheets, the author's rules will be used. Both reader and author rules override the web browser's default values.
- The imported style sheets also cascade with each other, in the order they are imported, according to the cascading rules defined below. Any rules specified in the style sheet itself override rules in imported style sheets. That is, imported style sheets are lower in the cascading order than rules in the style sheet itself. Imported style sheets can themselves import and override other style sheets, recursively.
- In CSS 1, all '@import' statements must occur at the start of a style sheet, before any declarations. This makes it easy to see that rules in the style sheet itself override rules in the imported style sheets.

3.1 Style Sheet Precedence Rules

This help to define the cascading order of CSS rules. In situations where conflict can occur between style sheets these precedence rules helps to determine the order in which the style sheets are implemented. There are four types of precedence:

- Rules marked "important" have the highest priority. This is rarely used. Example: H1{ colour:black !important; font-family:Times New Romans" }.
- Author rules have precedence over readers' rules. This means style sheet rules override browser preferences.

More specific rules have precedence over less specific rules. Examples:

- #foo { …} //ID selector highest priority
- P.big H1 { … } //Class higher over element
- P STRONG { … } //Two tags higher than single tag
- STRONG { …}

In case of tie, the last rule has priority

3.2 CSS Properties

Style sheets influence web document outlook by assigning values to style properties. Here we will discuss style properties and their respective list of possible values.

3.2.1 Notation for Property Values

The allowed values for each property are listed with syntax like the following:

- *Value:* N | NW | NE
- *Value:* [<length> | thick | thin]{1,4}
- *Value:* [<family-name> ,]* <family-name>
- *Value:* <url>? <colour> [/ <colour>]?
- *Value:* <url> || <colour>

The words between "<" and ">" give a type of value. This includes: <length>, <percentage>, <url>, <number>, and <colour>. Other words are keywords that must appear literally, without quotes the slash (/) and the comma (,) must also appear literally.

a. A bar (|): use to separate alternatives; one of them must be true;
b. A double bar (A||B): means that either A or B or both must occur;
   - Brackets ([]): use for grouping;

Every type, keyword, or bracketed group may be followed by one of the following modifiers:

a. An asterisk (*) indicates that the preceding type, word or group is repeated zero or more times.
b. A plus (+) indicates that the preceding type, word or group is repeated one or more times.
c. A question mark (?) indicates that the preceding type, word or group is optional.
d. A pair of numbers in curly braces ({A,B}) indicates that the preceding type, word or group is repeated at least A and at most B times.

1 Font properties

Setting font properties will be among the most common uses of style sheets.

Font-weight

Relative weight (boldness) of font

- normal | lighter | bold | bolder | 100 | 200 | … | 900
- Example:
- H1{ font-weight:200}
- H2 { font-weight:bolder}

font-style

- font face type within a family
- normal italic | oblique
- Example:
- P { font-style:normal

font-size

- Either relative or absolute size of font
- pt, pc, in, cm, mm | em, ex, px, % | xx-large | x-large | large | medium |small | x-small
- Example:
- P {font-size:14pt }
- SRTONG {font-size:80% }

font-family

- Typeface family for the font

Example:

- H1 { font-family:Arial }

2 Text properties

word-spacing

- Format: Value:normal |<length>
- The length unit indicates an addition to the default space between words. Value can be negative, but there may be implementation-specific limits.

Example: H1 { word-spacing:1em }.

- Here, the word-spacing between each word in ‘H1’ elements would be increased by ‘1em’
- The browser is free to select the exact spacing algorithm. Text-align property may also be added to word spacing.

letter-spacing

- Format: Value:normal |<length>
- Just like the word-spacing, the length unit indicates an addition to the default space between words. Value can be negative, but there may be implementation-specific limits. The browser is free to select the exact spacing algorithm. Text-align property may also be added to word spacing.

Example: BLACKQUOTE { letter-spacing:0.1em }.

- Here, the word-spacing between each word in 'H1' elements would be increased by '1em'
- With a value of 'normal', the UAs may change the space between letters to justify text. This will not happen if 'letter-spacing' is explicitly set to a <length> value.

text-decoration

- *Value:* none | [underline || overline || line-through || blink]
- This property describes decorations or additions that are added to the text of an element. If the element has no text (e.g. the 'IMG' element in HTML) or is an empty element (e.g. '<EM></EM>'), this property has no effect. A value of 'blink' causes the text to blink.
- The colour(s) required for the text decoration should be derived from the 'colour' property value.
- This property is not inherited, but elements should match their parent. E.g., if an element is underlined, the line should span the child elements. The colour of the underlining will remain the same even if descendant elements have different 'colour' values.
- A:link, A:visited, A:active { text-decoration: underline }. This example will underline the text of all links.

text-transform

- *Value:* capitalise | uppercase | lowercase | none
- 'capitalise': uppercases the first character of each word
- 'uppercase': uppercases all letters of the element
- 'lowercase': lowercases all letters of the element
- 'none': neutralises inherited value.

Example: H1 { text-transform: uppercase }

vertical-align

- Determines how elements are positioned vertically.
- *Value:* baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage>
- 'baseline'
- align the baseline of the element (or the bottom, if the element doesn't have a baseline) with the baseline of the parent
- 'middle'
- Align the vertical midpoint of the element (typically an image) with the baseline plus half the x-height of the parent
- 'sub'
- subscript the element
- 'super'
- superscript the element

- ‘text-top’
- align the top of the element with the top of the parent element's font
- ‘text-bottom’
- align the bottom of the element with the bottom of the parent element's font
- ‘%’
- Percentage values refer to the value of the 'line-height' property of the element itself.

text-align

- Determines how paragraphs are positioned horizontally
- Value:left | right | center | justify

text-indent

- specify the indentation of the first line of the paragraph.
- Value:<length> | <percentage>. The value of 'text-indent' may be negative, but there may be implementation-specific limits. Possible values of “length” and “percentage” include: +/-pt, in, cm, mm | +/-em, ex, px, %

Example: P {text-indent:-25px}

line-height

- specifies the distance between two consecutive baselines in a paragraph
- value:normal | number |pt, pc, in, cm, mm | em, ex, px, %

Example: .double { line-height:20% }

- DIV { line-height:1.5em }
- Having gone through font and text properties, now let us see how they work

Example 1:

Letter.html

- <html>
- <title>Application Letter</title>
- <LINK REL="STYLESHEET" HREF="Letter.CSS" TYPE="text/css">
- <head>
- <body>
- <P CLASS="rhead">
- May 1, 2009<HR>
- <P CLASS="rhead">
- 050020011<BR>
- Adeoye Joseph<BR>
- 400L Computer Science
- <P CLASS="1head">
- Director Academic Planning<BR>
- National Open University<BR>

- Victoria Island Lagos<BR>
- Nigeria.
- <P>
- <BR>
- Dear Sirs,
- <P CLASS ="body">

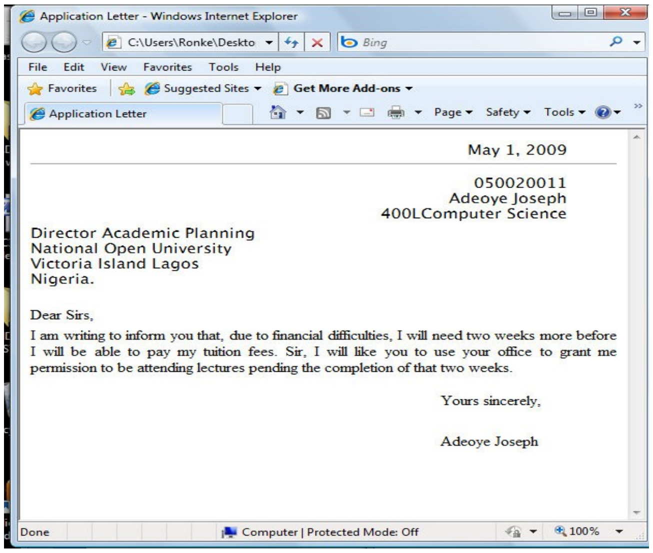
I am writing to inform you that, due to financial difficulties, I will need two weeks more before I will be able to pay my tuition fees. Sir, I will like you to use your office to grant me permission to be attending lectures pending the completion of that two weeks.<BR>

- <P CLASS ="foot">
- Yours sincerely,<BR>
- Adeoye Joseph
- </body>
- </html>

Letter.CSS

- p.{ margin-top:5px}
- P.rhead{text-align: right; margin-right: 0.5in; font-family: Lucida Sans}
- P.1head{font-family: Lucida Sans}
- P.body{ text-align:Justify; text-indext:0.5in}
- P.foot {margin-left:70%; line-height:300%}

The output is displayed below:


Application Letter - Windows Internet Explorer
C:\Users\Ronke\Deskto
Bing
File Edit View Favorites Tools Help
Favorites Suggested Sites Get More Add-ons
Application Letter
Page Safety Tools
Done Computer | Protected Mode: Off 100%

May 1, 2009

050020011
Adeoye Joseph
400LComputer Science

Director Academic Planning
National Open University
Victoria Island Lagos
Nigeria.

Dear Sirs,

I am writing to inform you that, due to financial difficulties, I will need two weeks more before I will be able to pay my tuition fees. Sir, I will like you to use your office to grant me permission to be attending lectures pending the completion of that two weeks.

Yours sincerely,

Adeoye Joseph

SELF-ASSESSMENT EXERCISE

Add title to the letter in **Example 1** above, underline it and justify it to the center with STRONG font- size 12.

3 Background property

colour

- colour the text or foreground colour
- value:colour name |#RRGGBB | #RGB | rgb(rrr, ggg, bbb) | rgb(rrr%, ggg%, bbb%)
- Example: P { colour: blue }
- H3 {colour: rgb(255, 0, 0) } /* red */

Background-image

- specifies an image to use as the background of region
- value:<url> | none
- example: Body { background-image: url (Bluce.gif);}

background-repeat

- specifies how to tile the image in the region
- *Value:* repeat | repeat-x | repeat-y | no-repeat

A value of 'repeat' means that the image is repeated both horizontally and vertically. The 'repeat-x' ('repeat-y') value makes the image repeat horizontally (vertically), to create a single band of images from one side to the other. With a value of 'no-repeat', the image is not repeated.

```
BODY {
        background: red url(pendant.gif);
        background-repeat: repeat-y;
            }
```

In the example above, the image will only be repeated vertically.

background-position

If a background image has been specified, the value of 'background-position' specifies its default position.

Value: [<percentage> | <length>]{1,2} | [top | center | bottom] || [left | center | right]
default: 0% 0%

With a value pair of '0% 0%', the upper left corner of the image is placed in the upper left corner of the box that surrounds the content of the element (i.e., not the box that surrounds the padding, border or margin). A value pair of '100% 100%' places the lower right corner of the image in the lower right corner of the element. With a value pair of '14% 84%', the point 14% across and 84% down the image is to be placed at the point 14% across and 84% down the element.

With a value pair of '2cm 2cm', the upper left corner of the image is placed 2cm to the right and 2cm below the upper left corner of the element.

If only one percentage or length value is given, it sets the horizontal position only, the vertical position will be 50%. If two values are given, the horizontal position comes first. Combinations of length and percentage values are allowed, e.g. '50% 2cm'. Negative positions are allowed.

One can also use keyword values to indicate the position of the background image. Keywords cannot be combined with percentage values, or length values. The possible combinations of keywords and their interpretations are as follows:

- 'top left' and 'left top' both mean the same as '0% 0%'.
- 'top', 'top center' and 'center top' mean the same as '50% 0%'.

Example:

- BODY {background: url(banner.jpeg) right top } /* 100% 0% */
- BODY {background: url(banner.jpeg) top center } /* 50% 0% */
- background

The 'background' property is a shorthand property for setting the individual background properties (i.e., 'background-colour', 'background-image', 'background-repeat', 'background-attachment' and 'background-position') at the same place in the style sheet.

Format: *Value:* <background-colour> || <background-image> || <background-repeat> || <background-attachment> || <background-position>

Example: Body {background: red }

SELF-ASSESSMENT EXERCISE

Using CSS code, place the logo of Open University as the background image in **Example 1** above.

4.0 CONCLUSION

This unit discusses some properties of CSS use to define the outlook of web pages such as text, font, box model, border, outline etc. These properties helps designer to control the look of their web pages. It also discusses the type of precedence that can be given to CSS rules.

5.0 SUMMARY

In this unit, we have learnt:

- the use of: Background, Text, Font, Box Model, font, Border Outline, Margin, Padding, List, and Table property
- using more than one style sheet for a document (cascade)
- how to create style sheet layer
- the style sheet precedence rules.

6.0 TUTOR-MARKED ASSIGNMENT

i. Develop a web page display names of 5 colleagues from your class using external style sheet. Hint [specify the heading to be H3, colour the horizontal line to be "blue" and insert the image of your favorite colleague].

ii. Using external CSS format, specify the background colour, font-size, font-family, text-align and margin of your output of tutor marked assignment in Unit 3.

7.0 REFERENCES/FURTHER READING

Michael Morrison (2001), " HTML and XML for beginners"

http://dev.opera.com/articles/view/4-the-web-standards-model-html-css-a/

http://www.corewebprogramming.com

UNIT 5 CASCADING STYLE SHEET PROPERTIES CONTINUE

CONTENTS

1.0 INTRODUCTION

This unit continues on cascading style sheet properties. Now we are going to discuss the Box Model property, Border property, Outline property, Margin, padding property, List property, and Table property.

2.0 OBJECTIVES

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

- describe the use of: box model property
- create outline property, margin and padding property, list and table property.

3.0 MAIN CONTENT

3.1 Box property

CSS assume that all elements result in one or more rectangular regions. This is referred to as bounding box. Styles can specify the margins, border, and padding of the bounding box. The size of the margin, border and padding are set using the following format:

- 'margin'
- *Value:* [<length> | <percentage> | auto]{1,4}

The 'margin' property is a shorthand property for setting 'margin-top', 'margin-right', 'margin-bottom' and 'margin-left' at the same place in the style sheet.
If four length values are specified they apply to top, right, bottom and left respectively.

If there is only one value, it applies to all sides,

If there are two or three, the missing values are taken from the opposite side.

For example: H1 {margin-top: 2em }, sets all margin to 2em.

A negative value is allowed, but there may be implementation-specific limits.

'padding-top'
Value: <length> | <percentage>

default value: 0;

Percentage values: refer to width of closest block-level ancestor

This property sets the top padding of an element.

Example: BLOCKQUOTE {padding-top: 0.3em }
- 'padding-bottom'
- *Value:* <length> | <percentage>
- *Initial:* 0
- *Percentage values:* refer to width of closest block-level ancestor

This property sets the bottom padding of an element.

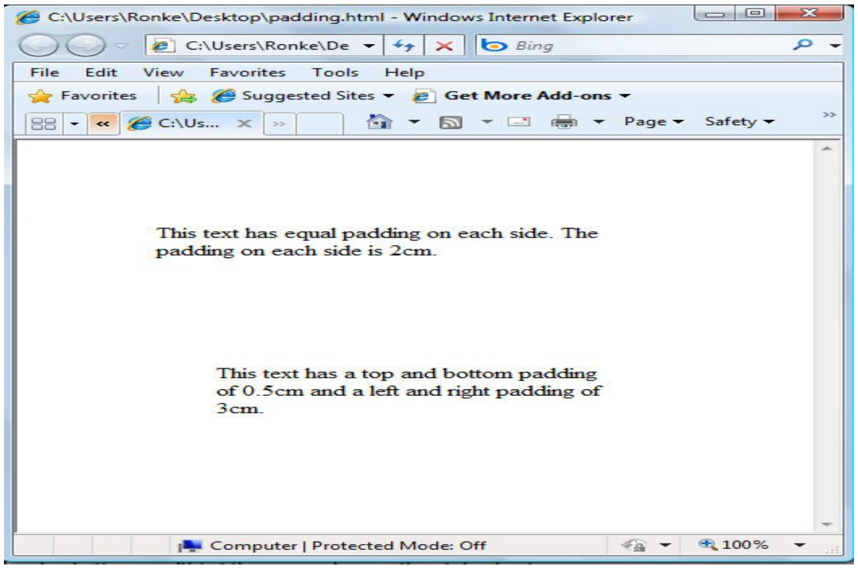
a. Using only ‘padding’ property is a short form of setting padding for top, bottom, left, and right at the same place in the style sheet.
b. If four values are specified, they applied to all the four sides: top, bottom, left, and right.
c. Example: H3 { background:blue; padding: 2em 2em }
d. The above example sets padding 2em vertically (top and bottom) and 2em horizontally (right and left).

SELF-ASSESSMENT EXERCISE

i. Open your text editor and type the following lines of codes.
   a. <html>
   b. <head>
   c. <style type="text/css">
   d. p.ex1 {padding:2cm}
   e. p.ex2 {padding:0.5cm 3cm}
   f. </style>
   g. </head>
   h. <body>

i. <p class="ex1">This text has equal padding on each side. The padding is 2cm.on each sides</p>
j. <p class="ex2">This text has a top and bottom padding of 0.5cm and a left and right padding of 3cm.</p>
k. </body>
l. </html>

This practice shows us, how the padding properties work and the output should be like the figure below.


C:\Users\Ronke\Desktop\padding.html - Windows Internet Explorer
File Edit View Favorites Tools Help
Favorites Suggested Sites Get More Add-ons
Page Safety
Computer | Protected Mode: Off
100%

This text has equal padding on each side. The padding on each side is 2cm.

This text has a top and bottom padding of 0.5cm and a left and right padding of 3cm.

- 'border-width'
- *Value:* thin | medium | thick | <length>
- *default value:* 'medium'

The 'border' property is a shorthand property for setting ‘border-right-width', ‘border-left-width' ‘border-top-width' ‘border-bottom-width' at the same place in the style sheet.

There can be from one to four values, with the following interpretation:

a. one value: all four border widths are set to that value;
b. two values: top and bottom border widths are set to the first value, right and left are set to the second;
c. three values: top is set to the first, right and left are set to the second, bottom is set to the third;
d. four values: top, right, bottom and left, respectively;

In the examples below, the comments indicate the resulting widths of the top, right, bottom and left borders:

- H1 { border-width: thin } /* thin thin thin thin */
- H1 { border-width: thin thick } /* thin thick thin thick */
- H1 { border-width: thin thick medium } /* thin thick medium thin */
- H1 { border-width: thin thick medium thin } /* thin thick medium thin */

NOTE: margin, padding, and border-width cannot be negative.

- 'border-colour'
- *Value:* <colour>{1,4}
- *default:* the value of the 'colour' property

The 'border-colour' property sets the colour of the four borders. 'border-colour' can have from one to four values, and the values are set on the different sides as for 'border-width' above.

If no colour value is specified, the value of the 'colour' property of the element itself will take its place:

- P {
- colour: black;
- background: white;
- border: solid;
- }
- 'border-style'
  - a. *Value:* none | dotted | dashed | solid | double | groove | ridge | inset | outset
  - b. *default:* none

The 'border-style' property sets the style of the four borders. It can have from one to four values, and the values are set on the different sides as for 'border-width' above.

- #xy34 {border-style: solid dotted }
- 'border'
  - a. *Value:* <border-width> || <border-style> || <colour>
  - b. *default:* not defined for shorthand properties

The 'border' property is a shorthand property for setting the same width, colour and style on all four borders (top, bottom, right, and left) of an element.

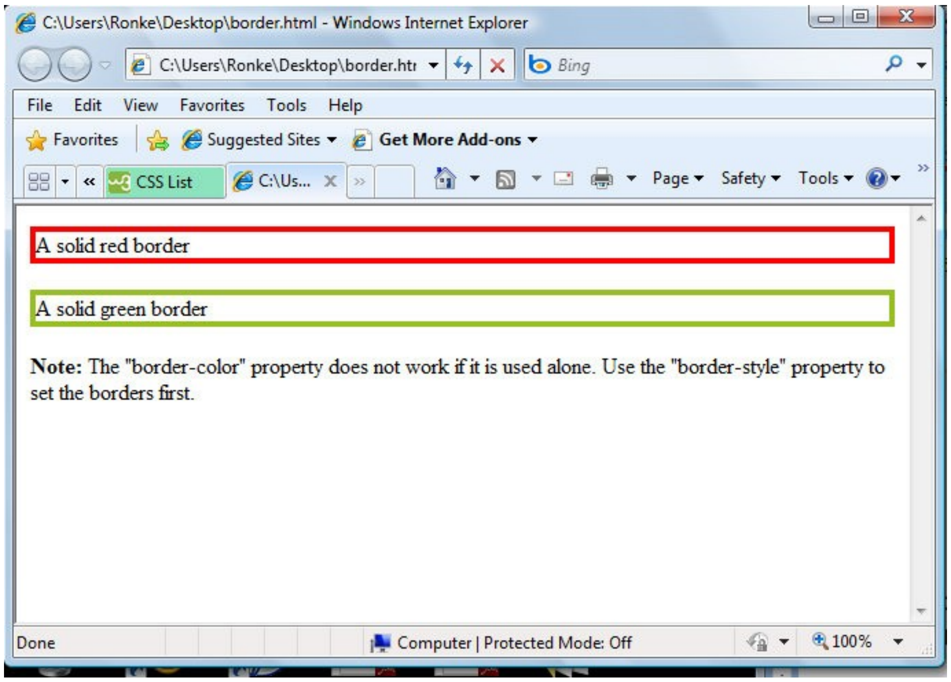
For example, P {border: solid red } is equivalent to the set of four rules shown below:

- P {
- border-top: solid red;
- border-right: solid red;
- border-bottom: solid red;
- border-left: solid red
- }

Example 2: In this example, we want to see how border property works. Open your note pad and type the following lines of code:

- <html>
- <head>
- <style type="text/css">
- p.one
- {
- border-style:solid;
- border-colour:red;
- }
- p.two
- {
- border-style:solid;
- border-colour:#98bf21;
- }
- </style>
- </head>
- <body>
- <p class="one">A solid red border</p>
- <p class="two">A solid green border</p>
- <p><b>Note:</b> The "border-colour" property does not work if it is used alone. Use the "border-style" property to set the borders first. </p>
- </body>
- </html>

The output is displayed below:


C:\Users\Ronke\Desktop\border.html - Windows Internet Explorer
File Edit View Favorites Tools Help
Done
Computer | Protected Mode: Off
100%

A solid red border

A solid green border

Note: The "border-color" property does not work if it is used alone. Use the "border-style" property to set the borders first.

Unlike the shorthand 'margin' and 'padding' properties, the 'border' property cannot set different values on the four borders. To do so, one or more of the other border properties must be used.

3.2 Classification Properties

These properties classify elements into categories more than they set specific visual parameters.

- **'display'**

a. *Value:* block | inline | list-item | none
b. *default:* block

This property describes how/if an element is displayed on the canvas (which may be on a printed page, a computer display etc.).

An element with a 'display' value of 'block' opens a new box. The box is positioned relative to adjacent boxes

Elements like 'H1' and 'P' are of type 'block'. A value of 'list-item' is similar to 'block' except that a list-item marker is added.

An element with a 'display' value of 'inline' results in a new inline box on the same line as the previous content. The box is dimensioned according to the formatted size of the content.

Example:

- P { display: block }
- EM { display: inline }
- LI { display: list-item }
- IMG { display: none }

The last rule turns off the display of images.

'list-style-type'

- *Value:* disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none
- Applies to elements with ‘display’ value ‘list-item’

This property is used to determine the appearance of the list-item marker if 'list-style-image' is 'none' or if the image pointed to by the URL cannot be displayed.

Example:

- OL { list-style-type: decimal } /* 1 2 3 4 5 etc. */
- OL { list-style-type: lower-alpha } /* a b c d e etc. */
- OL { list-style-type: lower-roman } /* i ii iii iv v etc. */

'list-style-image'

- *Value:* <url> | none
- *default:* none

This property sets the image that will be used as the list-item marker. When the image is available it will replace the marker set with the 'list-style-type' marker.

Example:
UL { list-style-image: url(http://png.com/ellipse.png) }
'list-style-position'
Value: inside | outside
Initial: outside
Applies to: elements with 'display' value 'list-item'

The value of 'list-style-position' determines how the list-item marker is drawn with regard to the content.

'list-style'

- *Value:* [disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none] || [inside | outside] || [<url> | none]
- *Applies to:* elements with 'display' value 'list-item'

- The 'list-style' property is a shorthand notation for setting the three properties 'list-style-type', 'list-style-image' and 'list-style-position' at the same place in the style sheet.
- UL { list-style: upper-roman inside }
- UL UL { list-style: circle outside }

SELF-ASSESSMENT EXERCISE

i. Modify the **Activity A** with border-style:solid, border-colour:red and border-style:solid, border-colour:#98bf21for p.ex1 and p.ex2 respectively.
ii. Using CSS list type property, develop a simple web page that shows the full name of ten (10) classmates.

4.0 CONCLUSION

This unit continues on cascading style sheet properties in the previous unit. The box model property, border property, outline property, margin, padding property, list property, and table property were discussed.

5.0 SUMMARY

In this unit, we have learnt:

- Use of: Box model property
- Outline property, margin and
- Padding property, List and Table property.

6.0 TUTOR-MARKED ASSIGNMENT

Create a web page named **HOME** linked to module two unit two. On the page:

i. Write a brief note welcoming customers to the site.
ii. Using appropriate list-style property, itemise the goods available for order.
iii. Create a heading “PHONEX ONLINE MOBILE PHONE SHOP”. Merge the heading to center with Solid red colour border.

7.0 REFERENCES/FURTHER READING

Michael Morrison (2001), " HTML and XML for beginners"

http://dev.opera.com/articles/view/4-the-web-standards-model-html-css-a/

http://www.corewebprogramming.com

MODULE 3 JAVASCRIPT AND VISUAL BASIC SCRIPT (VBSCRIPT)

UNIT 1 JAVASCRIPT

CONTENTS

1.0 INTRODUCTION

Scripting language are simple interpreted programming languages. Most popular among these scripting languages is JavaScript. JavaScript helps to make web pages more interactive. With JavaScript web pages are no longer static. Unlike Java programming, JavaScript is simpler and need not be complied.

2.0 OBJECTIVES

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

- identify where to put your scripts in an HTML document
- create how to hide scripts from old web browsers
- create how to put comments in scripts
- create how to display messages using alert, confirm, and prompt boxes
- create how to call functions with a button.

3.0 MAIN CONTENT

JavaScript Files

JavaScript is a scripting language that can be used on both client and server side. JavaScript add interactivity to Web pages. It assists in completion and checking of forms and also provides user control with client-side of page appearance and content.

3.1 Types of JavaScript

1 Client side - runs in browser
2 Server side - runs on server - embedded in web page but processed on server.

3.2 Placing Scripts

If the JavaScript code is short, it can be included in the HTML document.

1 in Head tag - this prevents user triggered errors during downloading
2 in body - including within body tag as an attribute
3 within forms as Event handlers - script tag not required

It can be also placed in a separate .js file

- To add clarity to an HTML document.
- To share JavaScript code across multiple HTML documents.
- To hide the script code.

In this case, viewer can only see the location of the script file but not the contents.

Scripts are embedded in pages between <script> </script>tags

```
Format <script type=”text/JavaScript”>
        Statement;
</script>
```

NOTE: src attribute can also be placed in the script tag but this is not supported by Internet Explorer.

3.3 Comment TAGS

JavaScript supports two types of comment:

- // for single line comment
- /* */ for multiple line comments

Adding comment to your script is very important so that you and other will know what various sections of code is used for.

3.4 Hiding JavaScripts from Old Browser

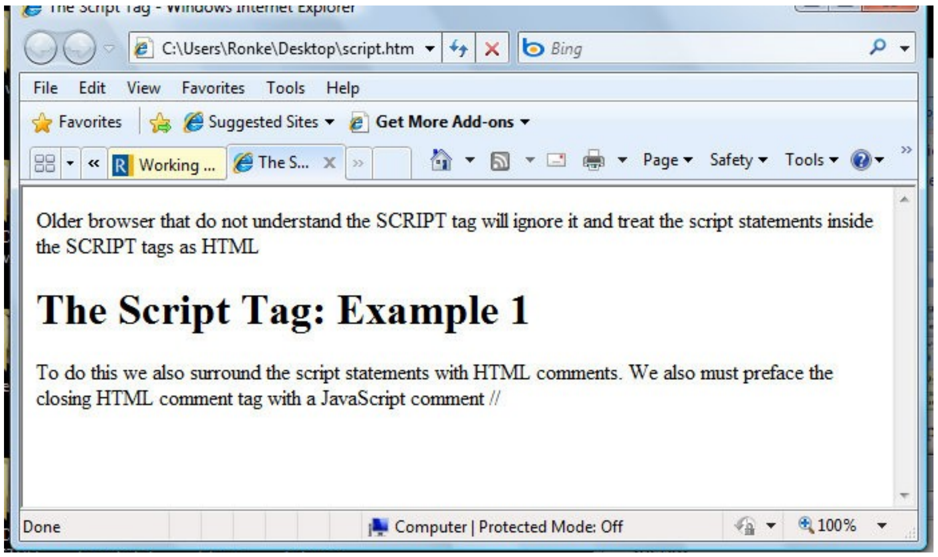
Browsers that were created before the script tag was invented will correctly ignore tags they do not understand. Such browsers will treat script language as ordinary plain text. To get around this, the script code can be written inside HTML comment tag (<!—this is a comment.-->). Let see how this is done.

Example 1:

- <HTML>
- <HEAD>
- <TITLE>The Scripting Language</TITLE>
- <SCRIPT Language = “JavaScript”>
- <!-- //Hide Script from non-script browsers
- document.write ("<P>Older browser that do not understand the SCRIPT tag will ignore it ")
- document.write ("and treat the script statements inside the SCRIPT tags as HTML</P>")
- // -->
- </SCRIPT>
- </HEAD>
- <BODY>
- <H1>The Script Tag: Example 3</H1>
- <P>
- <SCRIPT Language="JavaScript">
- <!--
- document.write("To do this we also surround the script statements ")
- document.write("with HTML comments. We also must preface the closing ")
- document.write("HTML comment tag with a JavaScript comment //")
- // -->
- </SCRIPT>
- </P>

- </BODY>

</HTML>


The Script Tag - Windows Internet Explorer
C:\Users\Ronke\Desktop\script.htm
Bing
File Edit View Favorites Tools Help
Favorites Suggested Sites Get More Add-ons
Working ... The S...
Page Safety Tools
Done Computer | Protected Mode: Off 100%

Older browser that do not understand the SCRIPT tag will ignore it and treat the script statements inside the SCRIPT tags as HTML

The Script Tag: Example 1

To do this we also surround the script statements with HTML comments. We also must preface the closing HTML comment tag with a JavaScript comment //

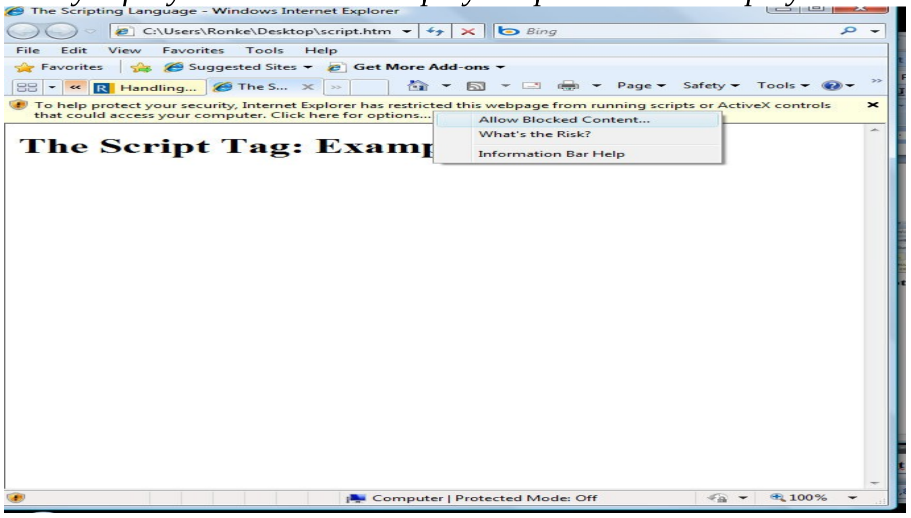
Fig.1.1: Example 1

SELF-ASSESSMENT EXERCISE

Create another file for **Example 1** above; remove the HTML comment tags that used to wrap the JavaScript. Run the code and compare the output with **Example1** above.

If your output is the same with **Example 1** output above, it means your browser is a recent version. So you can continue to run your script without putting the script statements in HTML comment tag.

NOTE:
If you are using Window Vista or latest Internet explorer like Explorer 7 and above, you will have to right click on "allow blocked content" and click yes for your browser to display script content as displayed below.


The Scripting Language - Windows Internet Explorer
C:\Users\Ronke\Desktop\script.htm
Bing
File Edit View Favorites Tools Help
Favorites Suggested Sites Get More Add-ons
Handling... The S...
Page Safety Tools
Allow Blocked Content...
What's the Risk?
Information Bar Help
Computer | Protected Mode: Off 100%

To help protect your security, Internet Explorer has restricted this webpage from running scripts or ActiveX controls that could access your computer. Click here for options...

The Script Tag: Examp

3.5 JavaScript Variables

JavaScript accepts the following types of variables:

- **Numeric:** Any numeric value, whether a whole number (an *integer*) or a number that includes a fractional part (a *real*), e.g., 23, 2.412
- **String:** collection of characters e.g. Titi, University.
- **Boolean:** a True or False

Note:
- JavaScript is not strongly types like java programming.
- Type of variables are not declared
- Variable names are case sensitive in JavaScript!
- Must start with letter or "_". Any combination of letters, numbers, "_"
- When a new variable is created or declared, it must be preceded by the word *var*
- No need to end statements with a semicolon
- New line is assumed to be a new statement
- Examples of JavaScript variables:
- var age = 25 (numeric)
- var surname = "Oyebade" (String)
- var married = True (Boolean)

3.6 Operators

Operators are types of command. They act on variables and/or literals and produce results.

1 **Binary Operators:** binary operators are operators that accept only **two** inputs. JavaScript support the following operators:
   - +Addition
   - Subtraction
   - / Division
   - Multiplication
   - % Modulus

2 **Unary Operators:** unary are operators that requires only **one** value as input
   - ++ Increment Increase value by 1
   - -- Decrement Decrease value by 1
   - - Negation Convert positive to negative, or vice versa

Examples:

- 5++ = 5 increase 5 by 1
- 6-- = 5 decrease 6 by 1
- -5 = -5 negate 5 so it becomes -5

3 **Comparison Operator:** As well as needing to assign values to variables, we sometime need to compare variables or literals. Comparison Operators compare two values and produce an output which is either true or false. Different from the assignment operator, the comparison operator tests to see if variables (values) are *already* equal to one another. The following are comparison operators supported by Jascript:

- == (two equals signs): it means 'is equal to'.
- e.g. registeredStudents == totalStudents
- != (an exclamation mark followed by equals sign): it means 'is NOT equal to'.
- e.g. registeredStudents != totalStudents
- < means 'less than'
  means 'greater than'
- <= means 'less than or equal to'
- >= means 'greater than or equal to'

4 **Assignment operator (=):** which means 'becomes equal to. The assignment operator *makes* two things equal to one another.

5 **Combined Operators:** we can also combine these operators and the assignment operator in following ways:

- + = 'becomes equal to itself plus'
- -= 'becomes equal to itself minus'
- *= 'becomes equal to itself multiplied by'
- /= 'becomes equal to itself divided by'
- %= 'becomes equal to the amount which is left when it is divided by'

Example:
total += Sum
Means:
total = total + sum

6 **Logical Operators:** these operators are used to combine results of other conditional tests. They are:

&&	Logical AND
| |	Logical OR

Example:
if (x > 2 && x < 10)
Placing the && between the two conditions means that if statement will only be carried out if BOTH conditions are true. If only one of the conditions is true (e.g., x is greater than 0 but also greater than 10) the condition will return false and the if statement won't be carried out.

7. **Concatenation operator (+):** the character concatenation operator + is used to join text, number and characters. This is commonly used when you have to combine long sentence or words taking from different input fields.

3.7 JavaScript Control Flow

The power of programming languages comes from their ability to respond in different ways depending upon the data they are given. Every programming language that supports dynamic content includes statements which allow 'decisions' based on data given. JavaScript supports loop and conditional construct

I. **The *for* Loop:** it allows you to carry out a particular operation a fixed number of times. The *for* is controlled by setting three values:

- an initial value
- a final value
- an increment

Format:

```
For (initial value; final value; increment)
{
        Statements(s);
}
```

Example:

```
for (x=5; x>=0; x--)
{
        alert('x = ' + x);
}
```

2 **If …Else statement:** it makes decision to be made between alternative choices based on one condition. The If-Else statement in JavaScript has the following syntax:

```
if (condition)
{
        statement;
        Statement
}
Else
{
        statement;
        Statement
};
```

The condition is the information on which we are basing the decision. The browser first interpret the condition, if it is true, the browser will carry out the statements within the if... section; if the condition is false it will carry out the statements within the else... section.
NOTE: Always observe the positioning of the semi-colons.
If you are using both the if... and the else... parts of the statement, it is important NOT to put a semi-colon at the end of the if... part. If you do, the else... part of the statement will never be used.
A semi-colon is normally placed at the very end of the if...else... statement, although this is not needed if it is the last or only statement in a function.

3 **The While Loop:** The *while* loop like the *for* loop allow you to carry out a particular operation a number of times.
Format:

```
While (condition)
{
        Statement(s);
}
```

Example:

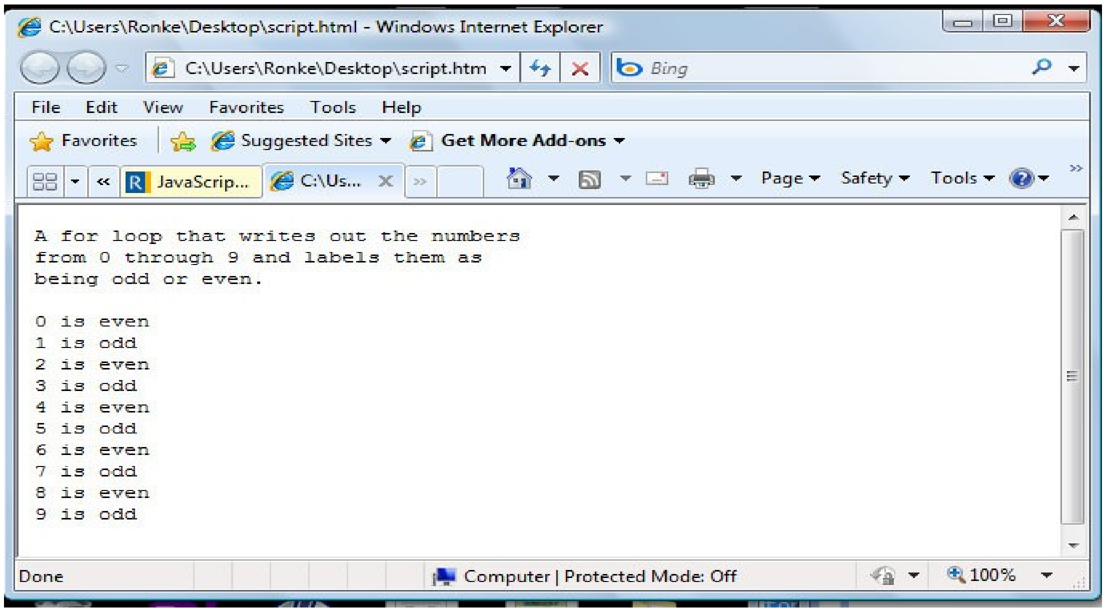
```
var x = 30;
alert (“Count …”);
while (x > 0)
{
x--;
};
Alert (“Finished!”);
```

Here, x is initially set to a high value (30). It is then reduced by one each time through the loop using the decrement operator (x--). So long as x is greater than zero the loop will continue to operate, but as soon as x reaches zero the loop condition (x > 0) will cease to be true and the loop will end.

Example 2: Open your notepad and type the following lines of code. At the end of this practice, you will practically see how loops work in JavaScript.

- <HTML>
- <HEAD>
- <TITLE>Testing JavaScript Loop Statement</TITLE>
- </HEAD>
- <BODY>
- <PRE>
- A *for loop* that writes out the numbers from 0 through 9 and labels them as being odd or even.

- <SCRIPT Language="JavaScript">
- <!--
- for (i=0; i < 10; i++) {
- if (i%2 != 0)
- document.writeln(i + " is odd")
- else
- document.writeln(i + " is even")
- }
- // -->
- </SCRIPT>
- </PRE>
- </BODY>
- </HTML>


C:\Users\Ronke\Desktop\script.html - Windows Internet Explorer
C:\Users\Ronke\Desktop\script.htm
Bing
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Get More Add-ons
JavaScrip...
C:\Us...
Page Safety Tools
Done
Computer | Protected Mode: Off
100%

```
A for loop that writes out the numbers
from 0 through 9 and labels them as
being odd or even.

0 is even
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
```

Fig.1.2: Example2 Output

SELF-ASSESSMENT EXERCISE

Modify the script in example two to write out only the even numbers from 0 through 9.

4.0 CONCLUSION

Scripting language are interpreted programming languages. JavaScript is the most popular among the scripting languages. It makes web pages more interactive and dynamic.

5.0 SUMMARY

In this unit, we have learnt:

- where to put your scripts in an HTML document
- how to hide scripts from old web browsers
- how to put comments in scripts
- how to display messages using alert, confirm, and prompt boxes
- calling functions with a button.

6.0 TUTOR-MARKED ASSIGNMENT

Carry out example 2 using **While loop** statement for both odd and even numbers.

7.0 REFERENCES/FURTHER READING

Brooks & David, R. (2007). An Introduction to HTML and JavaScript for Scientists and Engineers.

David, F. (1998). *JavaScript*: *The Definitive Guide*. (3rd ed.). O'Reilly Media.

Larry, R. (2002).“Programming the Web Using XHTML and JavaScript”, McGraw-Hill.

UNIT 2 JAVASCRIPT AND OBJECT ORIENTED PROGRAMMING (OOP)

CONTENTS

1.0 INTRODUCTION

One important characteristic of JavaScript is its ability to work with objects. Objects are convenient means to package information and methods of working with this information.

Objects are important tools. There advantages include:

- Breaking down difficult programming problems into simpler problems that can solved by inventing different types of objects to solve the smaller parts of the problem
- Grouping or collection of data and ways of working with data into logically cohesive objects that do not interfere with other parts of your script. This allows you to work on small separate pieces that work together in simple and well defined ways rather than one large program all at once
- Objects can be reused or modified easily for reuse allowing programmers to avoid rewriting too much code from scratch when a new program is needed.

2.0 OBJECTIVES

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

- define the meaning of Object-Oriented Programming
- create and initialise objects in JavaScript
- create JavaScript forms and functions
- create event handling in JavaScript
- create getting and Setting Text Field in JavaScript.

3.0 MAIN CONTENT

What is Object- Oriented Programming?

The act of creating objects that collaborate to solve a programming problem is called object-oriented programming (OOP). OOP concept has been around for a while and has constituted an important way to construct programs. JavaScript is not designed to force you to use object-oriented design techniques but does make it possible to use many object-oriented design methods.

In order to access or change the attributes and contents of a Web page, JavaScript provides a convenient object called the document object. According to Netscape, it "contains information on the current document, and provides methods for displaying HTML output to the user."

In the Web browser, certain parts of the document are considered as objects. This includes the browser window, forms, buttons, text boxes, etc. JavaScript facilitates accessing and modifying objects in a document:

Format:
document.write()document
An application object that contains information about the document loaded into a window. In the JavaScript documentation the information about an object is referred to as its properties.

The dot between the document object and the write() method is significant. The dot operator allows you to access information (object properties) and methods for working with that information in an object. The method or property of an object is on the right side of the dot and the object is on the left side. For example *document.bgColour* is use to set background colour of a document.

write() write() is a method (also called a function) belonging to the document object. write() will output a string of text to the document. Any JavaScript expression can be placed inside the round brackets. The expression will be evaluated and the result written into the Web page.

3.1 Creating and Initialising Objects

3.1.1 Creating an Object

To create your own JavaScript objects you need to:

- call a function using the key word: new
- either write a special "constructor" function or call the Object(), Array(), Date() or other predefined constructor function.

Why writing your own function, it is often useful to create an object with a standard set of properties and methods. Because the function adds one set of properties and methods it in effect constructs an object of a certain type. If a particular type of object is wanted a constructor function can be designed to create it as needed. For example suppose we want to create an online quiz. A nice approach to this problem is to create a series of question objects that contain properties that include the question and the correct answer. Rather than creating each object and then adding properties one-by-one, we can pass the property values into the question function:

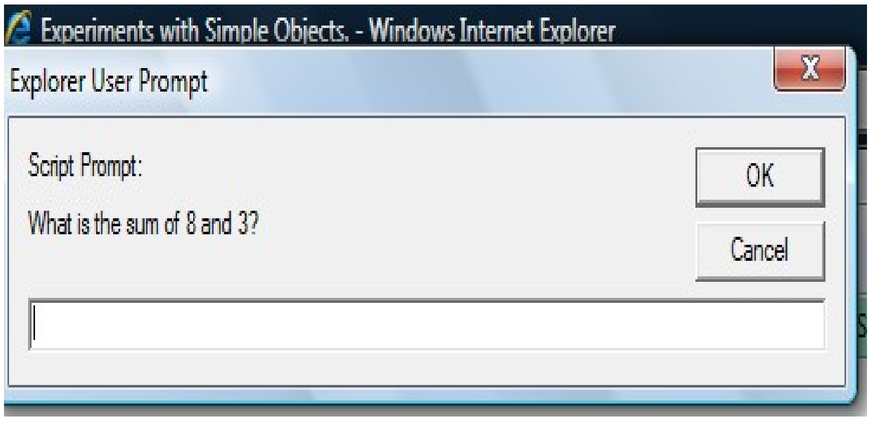
- function Question(question, answer){
- this.question = question
- this.answer   = answer
- }

The Question = new Question("What is the sum of 5 and 3?", 8)
In this example the function named Question() expects two parameters: question and answer. These are assigned to the properties this.question and this.answer.

The keyword **this** refers to the object that has just been created by the new operator. The properties of the object are created and receive values in the two assignment statements in the Question() function.

Example 1
- <HTML>
- <HEAD>
- <TITLE>Experiments with Simple Objects.</TITLE>
- <SCRIPT Language="JavaScript">

- <!--
- function Question(question, answer){
- this.question = question
- this.answer   = answer
- }
- q1 = new Question("What is the sum of 5 and 3?", 8)
- q2 = new Question("What is the sum of 8 and 3?", 11)
- userAnswer = prompt(q1.question, "")
- if (userAnswer == q1.answer){
- alert("Correct")
- }
- else {
- alert("Not Correct")
- }
- userAnswer = prompt(q2.question, "")
- if (userAnswer == q2.answer){
- alert("Correct")
- }
- else {
- alert("Not Correct")
- }
- //-->
- </SCRIPT>
- </HEAD>
- <BODY>
- </BODY>
- </HTML>


Experiments with Simple Objects. - Windows Internet Explorer
Explorer User Prompt
OK
Cancel

Script Prompt:

What is the sum of 8 and 3?

Fig.2.1: **Example 1**

Brief Explanation

In this practice, a Question object is constructed using the Question() function; a hint property is added to the Question object; a reference to the question object is passed to the showObjectProperties() function that displays all the properties and values of the object.

3.2 Functions in JavaScript

3.2.1 JavaScript Standard Function

- JavaScript two provides predefined functions: parseInt() and parseFloat().
- parseInt():converts, where possible, a string to a number.

Example:

- result = parseInt("2.14 Welcome")

This statement shows how parseInt() can be used. In this example, the string "2.14 Welcome" will be checked by the parseInt() function to see if it begins with text that can be converted to an integer. So the number 4 will be assigned to the variable result in this case.

parseFloat(): convert a string to a number that may include decimal number

Example:

- result = parseFloat("2.14 Welcome"). This works exactly like parseInt but will return 2.14 as the value of result because it allows fractional number.

NOTE: If either function does not find text it can convert to a number at the beginning of the string it returns a special number value NaN that means "not-a-number."

3.2.2 User- Defined Functions

The parseInt() and parseFloat() are JavaScript prewritten functions and are designed for their specific purpose. JavaScript also have the capability that allows users to create new functions within scripts. These functions are named segments of JavaScript statements. They usually have a single purpose, such as performing a complex calculation or verifying the data entered into an HTML form. Functions can be passed copies of variables or references to objects to work with.

User defined functions can be placed inside script tags anywhere in the document. However, they should be placed in the head of the document to guarantee they are loaded before being called from script statements in the body of the document.

Format:

- function functionName(parameter_1, parameter_2, .. parameter_n)
- {
- statement1
- statement2
- statementn
- }

NOTE:
The keyword function precedes the name of the function.

A function name is like a variable name in (it must start with a character and must not be the same as a JavaScript key word).

The parameter list is a comma separated list of variable names inside round brackets immediately following the function name.

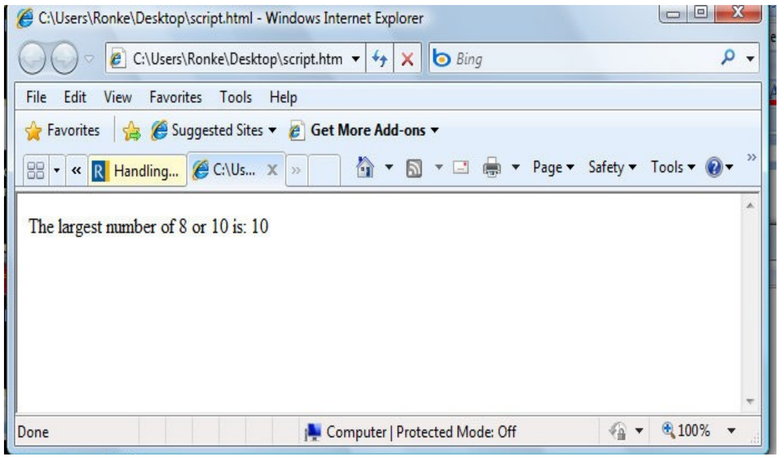
The statements to be executed when the function is called are contained inside the function body and are contained within curly braces.

Example 2: this example demonstrates the use of function block and if…else statement.

Open your notepad and type the following code:

- <HTML>
- <HEAD>
- <TITLE></TITLE>
- <SCRIPT Language="JavaScript">
- <!--

```
function maxNum(a, b){
  if (a < b)
    return b
  else
    return a
}
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
<!--
document.write("<P>")
document.write("The largest number of 8 or 10 is: " + maxNum(8, 10))
document.write("</P>")

// -->
</SCRIPT>
</BODY>
</HTML>
```


C:\Users\Ronke\Desktop\script.html - Windows Internet Explorer
C:\Users\Ronke\Desktop\script.htm
Bing
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Get More Add-ons
Handling...
C:\Us...
Page
Safety
Tools
Done
Computer | Protected Mode: Off
100%

```
The largest number of 8 or 10 is: 10
```

Fig.2.2: Output of Example 2

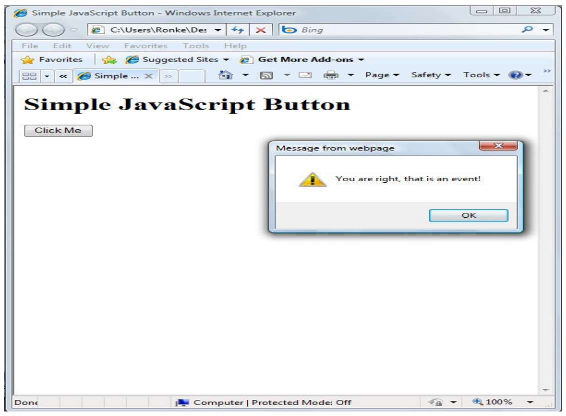
3.3 JavaScript Events

- Events are actions occurring as a result of users' interaction with web page.
- Keyboard and mouse events
- Capturing a mouse event is simple
- Load events
- The page first appears on the screen: "Loads", leaves: "Unloads"
- Form-related events
- onFocus() refers to placing the cursor onto a certain element.
- Others
- Errors, window resizing, etc

Example 3

- <html><head>
- <title>Simple JavaScript Button</title>
- <script language="JavaScript">
- <!--
- function ClickMe() {
- alert("You are right, that is an event!");
- }
- -->
- </script>

- </head>
- <body>
- <h1>Simple JavaScript Button</h1>
- <form>
- <input type="button" value="Click Me" onClick="ClickMe()">
- </form>
- </body></html>
- Output



Simple JavaScript Button

You are right, that is an event!

Fig.2.3: Example 3

SELF-ASSESSMENT EXERCISE

Using JavaScript function, develop a simple web page to compute circumference of a circle

3.4 JavaScript Form and Functions

JavaScript extends the functionality of HTML forms. It allows writing code that can check user input and display additional information to the user. This is achieved in two ways:

a. First, declaration of named functions much like in C/C++/Java. These functions are designed to perform specific task such as checking user input for errors, are called when certain events occur. Examples of event includes: user pressing a submit button in an HTML form or merely placing the cursor over a hypertext link.

b. Second, special HTML tag attributes called event handlers are placed in HTML tags. When an event occurs that the attribute is designed to handle, a short script in the handler is executed. Event handlers contain statement(s) that just call a function or two.

NOTE: To achieve this, you must both define the function to correctly get information from the form and place the call to the function in an event handler in an HTML form.

In previous practice we saw how JavaScript used function to display information by through ordinary clicking of a button. Now let see how to work with accepting user input

3.5 Event- Handling

Events occur when the user does something with the browser. This includes placing or moving mouse over a hypertext link, clicking a radio button in a form, clicking a check button in a form, or loading a page into a window. These events can trigger JavaScript code, giving browser the ability to respond to user actions when they occur. Events that occur when a user manipulates an HTML form are normally handled using HTML event handler attributes. These attributes are an extension to HTML that provides a method for the browser to invoke JavaScript code when the events they refer to occur.

Format: <HTMLTAG eventHandler="statement1; statement2; statementn">

NOTE: the semicolons separate the different JavaScript statements allowing more than one to be placed in the same line.

So far we have only being placing placed JavaScript inside <SCRIPT></SCRIPT> tags. JavaScript statements can also be placed in the quotes following an event handler attribute. Here are the meanings of each part of the tag:

HTMLTAG	HTML tags including: FRAMESET, BODY, FORM, SELECT, INPUT tags such as text, radio, checkbox, password, submit, reset, and button, and A.
eventHandler	Any of: onBlur, onClick, onChange, onFocus, onLoad, onMouseOver, onMouseOut, onSelect, onSubmit, onUnload, etc.
“statement1; ….”	JavaScript statements. Substitute single quotes for where you would normally put double quotes. Most often there are one or two statements, one of which is a function call.

Example: <INPUT TYPE=”Button” VALUE=”ClickMe” onClick=”clickIt(this.form)”>

The statement in quote in the onClick is executed when someone clicks the button.

The function clickIt is called when the user clicks the button.**this** refers to the current JavaScript object that reflects the tag, in this case a button object. **this.form** refers to the form the button is in. clickIt is being passed a reference to the object representing the form the button is in.

3.6 Getting and Setting Text Field

Getting and setting the text in an input text field is done by using the value property of the input field. For instance, the text of an input field named firstName in a form named stuName can be accessed like this:

- **result = document.stuName.firstName.value**

Similarly text can be placed in the field this way:

- **document.stuName.firstName.value = "Titi"**

It is unusual to see this form in a function designed to perform some operation with a form. Most often a reference to a form object, an input object, or a text value is passed to a function from the event handler that calls it. This is what happens in this example. A function is called by the onClick="" handler in the input button tag:

- **onClick="showAndClearField(this.form)"**

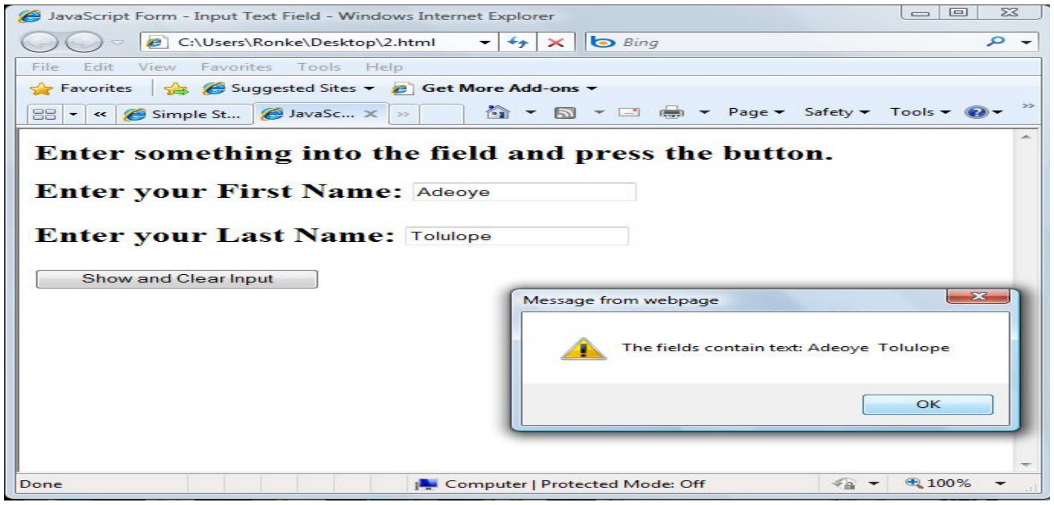
The keyword this refers to the input tag. For convenience each input object has a property named form that refers to the form it is in. Therefore this.form holds a reference to the form object and that is what is passed to the function.

Example 4

- <HTML>
- <HEAD>
- <TITLE>JavaScript Form - Input Text Field</TITLE>
- <SCRIPT Language="JavaScript">
- <!--//
- function showAndClearField(frm)
- {
- if (frm.firstName.value =="" && frm.lastName.value=="")
- alert("Hey! You didn't enter anything!")
- else

- alert("The fields contain text: " + frm.firstName.value + " " + frm.lastName.value)
- frm.firstName.value = ""; frm.lastName.value = ""
- }
- //-->
- </SCRIPT>
- </HEAD>
- <BODY>
- <FORM NAME="stuName">
- <H2>Enter something into the field and press the button. <br>
- <P>Enter your First Name: <INPUT TYPE="TEXT" NAME="firstName"><BR><BR>
- Enter your Last Name: <INPUT TYPE="TEXT" NAME="lastName"><BR><BR>
- <INPUT TYPE="Button" Value="Show and Clear Input" onClick="showAndClearField(this.form)">
- </P>
- </FORM>
- </BODY>
- </HTML>

NOTE: In this practice function is passed a reference to the form. Since the parameter list of the function contains a variable named frm the reference to the form will be copied into frm. The data in the text field can now be read using:
frm.firstName.value


JavaScript Form - Input Text Field - Windows Internet Explorer

C:\Users\Ronke\Desktop\2.html

Enter something into the field and press the button.

Enter your First Name: Adeoye

Enter your Last Name: Tolulope

Message from webpage

The fields contain text: Adeoye Tolulope

Fig.2.4: Example 4

function showAndClearField(frm)
This function is called when the button in the form is pushed. The **frm** parameter is supposed to receive a reference to the form. The reference in this case is equivalent to **document.stuName** which is another way to refer to the form - as a property of the document object. The function is designed to report what is in the text field and then clear it.

if (frm.firstName.value == "")
This line checks to see if the text input field has been left empty. **frm.firstName** refers to the input text field named **firstName** in the form. **frm.firstName.value** refers to the value in the field.

alert("The field contains the text: " + frm.firstName.value)
Here **frm.firstName.value** also retrieves the value that has been entered into the firstName input field. In this case it is concatenated to another string and displayed in an alert dialog box.

frm.firstName.value = ""
Near the end of this short function the field is cleared by setting its value to an empty string.

<FORM NAME="stuName">
Since this example never uses **document.stuName** to access the form it is not really necessary to actually name the form. However, it is a good practice to do this.

<INPUT TYPE="TEXT" NAME="firstName">
<INPUT TYPE="TEXT" NAME="lastName">
In order to refer to the input text field the HTML tag that creates it must have a name attribute.

<INPUT TYPE="Button" Value="Show and Clear Input" onClick="showAndClearField(this.form)">

The button tag contains the **onClick=""** event handler. As a result whenever the button is pressed (or "clicked") the JavaScript inside the quotes is executed. In this case the function named **showAndClearField** is called and it passed the parameter **this.form**. Whenever you see the word **this** used in an event handler in an HTML tag the word **this** refers to the JavaScript object that reflects (or represents) the HTML object created by the tag. In this case **this** refers to the button. Every form element also has a property that refers to the form it is in. So **this.form** is a reference to the form object the button is in. Thus we are passing a reference to the form to the function.

3.7 JavaScript Frame

The default Window object contains a frames property holding an array of frames (other Window objects) contained by the current window or frame.

It also has parent and top properties referring to the directly enclosing frame or window and the top-level window, respectively

All of the properties of Window can be applied to any of these entries.

SELF-ASSESSMENT EXERCISE

Using JavaScript form and function to create a simple web page that takes course code as input and display the course unit in return.

4.0 CONCLUSION

JavaScript has the ability to work with objects which are convenient ways to package information and methods of working with this information.

5.0 SUMMARY

At the end of this unit, we have discussed the following:

- The meaning of Object-Oriented Programming, Creating and Initialising Objects in JavaScript, JavaScript Forms and Functions, Event Handling in JavaScript and Getting and Setting Text Field in JavaScript.

6.0 TUTOR-MARKED ASSIGNMENT

Using JavaScript Form and Function, develop an order processing web application for **PHONEX ONLINE MOBILE PHONE SHOP** in module 2 unit 2.

7.0 REFERENCES/FURTHER READING

Brooks & David, R. (2007). “An Introduction to HTML and JavaScript for Scientists and Engineers.”

David, F. (1998). “JavaScript: The Definitive Guide” (3rd ed.). O'Reilly Media.

Larry, R. (2002).“Programming the Web using XHTML and JavaScript”, McGraw-Hill.

UNIT 3 VISUAL BASIC SCRIPT

CONTENTS

1.0 INTRODUCTION

VBScript is a scripting language, like JavaScript it was designed as an extension to HTML. VB is a light version of Microsoft’s programming language Visual Basic. The web browser receives scripts along with the rest of the web document. It is the browser's responsibility to parse and process the scripts. Just like JavaScript, VBScript requires a little understanding of HTML.

2.0 OBJECTIVES

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

- define VBScript Data types
- describe VBScript Variables
- identify VBScript Operators
- explain VBScript Program Control statements.

3.0 MAIN CONTENT

Placing VBScript in an HTML Document

When a VBScript is inserted into a HTML document, the Internet browser will read the HTML and interpret the VBScript. The VBScript can be executed immediately, or at a later event.

- Head section: Usually all the "functions" are placed in the head section. The reason for this is to be sure that the script is loaded before the function is called.
- Body section: Scripts in the body section are executed when the page is loading.

Example of script in the Head section

- <html>
- <head>
- <script type="text/VBScript">
- some statements
- </script>
- </head>
- Example of script in the body section
- <html>
- <head>
- </head>
- <body>
- <script type="text/VBScript">
- some statements
- </script>
- </body>
- Script in both body and the head section
- <html>
- <head>
- <script type="text/VBScript">
- some statements
- </script>
- </head>
- <body>
- <script type="text/VBScript">
- some statements
- </script>
- </body>

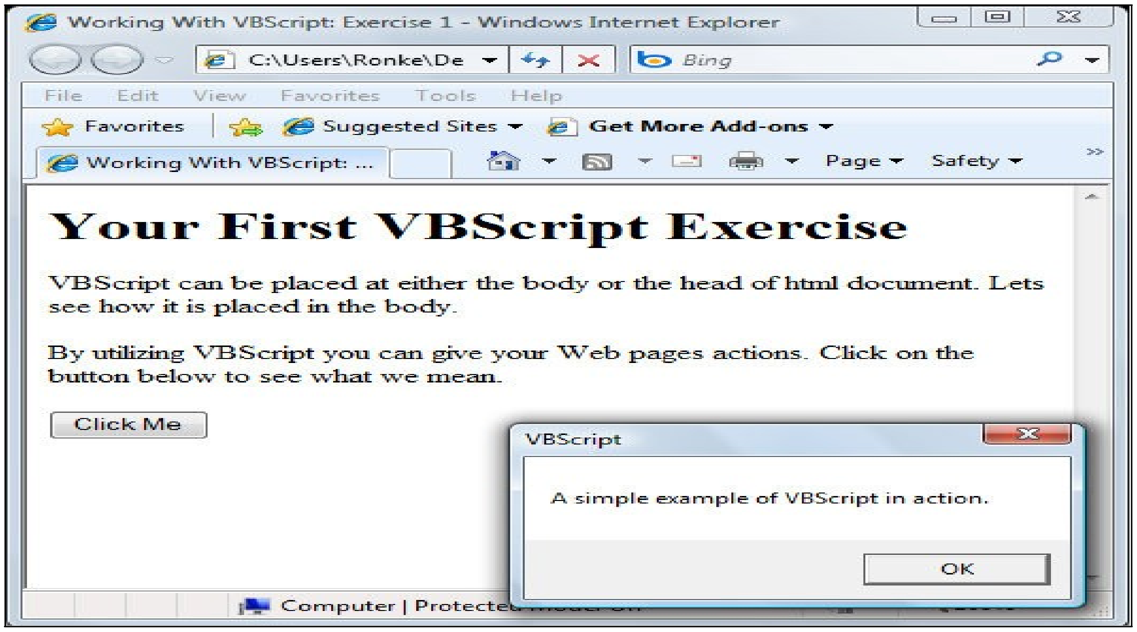
3.1 Where to Put the VBScript

Scripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event.

Let us see an example of using VBScript in the body of HTML documents

Example 1: Open a notepad and type the following lines of code:

- <HTML>
- <HEAD>
- <TITLE>Working with VBScript: Exercise 1</TITLE>
- </HEAD>
- <BODY>
- <H1>Your First VBScript Exercise</H1>
- <P> VBScript can be placed at either the body or the head of html document. Let's see how it is placed in the body.
- <p> By utilising VBScript you can give your Web pages actions.
- Click on the button below to see what we mean. </P>
- <FORM NAME="frmExercise1">
- <INPUT TYPE="Button" NAME="cmdClickMe" VALUE="Cli ck Me">
- <SCRIPT FOR="cmdClickMe" EVENT="onClick" LANGUAG E="VBScript">
- MsgBox "A simple example of VBScript in action."
- </SCRIPT>
- </FORM>
- </BODY>
- </HTML>


Working With VBScript: Exercise 1 - Windows Internet Explorer
C:\Users\Ronke\De
Bing
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Get More Add-ons
Working With VBScript: ...
Page
Safety
Click Me
VBScript
OK
Computer | Protected Mode: On

Your First VBScript Exercise

VBScript can be placed at either the body or the head of html document. Lets see how it is placed in the body.

By utilizing VBScript you can give your Web pages actions. Click on the button below to see what we mean.

A simple example of VBScript in action.

Fig.3.1: Example 1

Brief Explanation
We will explain these three lines of code:

Line

a. 1:<SCRIPT FOR="cmdClickMe" EVENT="onClick" LANGUA GE="VBScript">
b. Line 2: MsgBox "A simple example of VBScript in action."
</SCRIPT>

1. The first line defines a script.
The *FOR* argument specifies that this script is for the button named *cmdClickMe*, the name we have given our command button with the HTML <*INPUT*> tag.
The *EVENT* argument says that this script should be run when the button is clicked.
The *LANGUAGE* argument states that this is a VBScript module.
2. The second line is the only line of VBScript in this HTML document.
The *MsgBox* function simply displays a message dialog. We shall discuss more on this later.
3. The third line marks the end of the script.
In this practice, we simply inserted the VBScript module right after the HTML tag that defined the command button. While this method is functional, it is not the recommended approach. A better approach is to place all your script together within the HTML document. This will not make your HTML document more complex to read.

Preferred Method to Include VBScript

SELF-ASSESSMENT EXERCISE

i. Re-open the HTML document that you created in Example 1 and remove the three VBScript lines i.e.:
    a. <SCRIPT FOR="cmdClickMe" EVENT="onClick" LAN GUAGE="VBScript">
    b. MsgBox "A simple example of VBScript in action."
    c. </SCRIPT>
    d. Then modify the document by adding the scripting lines as shown below:
    e. <HTML>
    f. <HEAD>
    g. <TITLE>Working With VBScript: Exercise 1</TITLE>
    h. <SCRIPT LANGUAGE="VBScript">
    i. <!-- Instruct non-
    IE browsers to skip over VBScript modules.
    j. Sub cmdClickMe_OnClick
    k. MsgBox "A simple example of VBScript in action."
    l. End Sub
    m. -->
    n. </SCRIPT>
    o. </HEAD>
    p. <BODY>
    q. <H1>Your First VBScript Exercise</H1>
    r. <P> By utilising VBScript you can give your Web pages a ctions.
    s. Click on the button below to see what we mean. </P>
    t. <FORM NAME="frmExercise1">
    u. <INPUT TYPE="Button" NAME="cmdClickMe" VALU E="Click Me">
    v. </FORM>
    w. </BODY>
    x. </HTML>

ii. Then, run the script program and compare the output with the one created in Example 1.

3.2 VBScript Data Type, Variables, and Operators

3.2.1 Data Types

Most languages define types and sizes of number, characters, strings, float, decimal as distinct data types. But VBScript treats all data as one of two fundamental types: numbers or strings. Most of the time VBScript figures out how to make the conversions based on context.

When you need explicit control over data types VBScript has conversion functions and data type functions that return the type of variables and expressions.

3.2.2 Variables

Variables are used to store data items that can change (e.g. "vary") during the lifetime of a program. The data is actually stored in numbered memory addresses in the computer's Random Access Memory (RAM), however variables in high-level languages like VBScript provide convenient names for the memory addresses. In VBScript, all variables have one variable type called *variant*.

3.2.2.1 Declaring Variables

Variables in VBScript do not have to be "declared" before they are used, they spring to life when you first use them. This seems convenient, but it is now generally considered a bad idea, it mostly used in older VBScript version.

However, VBScript supports two methods for declaring variables; explicitly and implicitly.

1 The **Dim** statement allows you to explicitly declare the variables you intend to use. The **Option Explicit** is also supported, which *forces* you to use the **Dim** statement for all variables. When used, **Option Explicit** must be the first executable statement in the program.
Example: Dim Name
This statement declares the variable Student. You can also declare multiple variables on one line for example:
Dim Name, Department, Program, Level
NOTE: it is preferable to declare each variable separately.

2 Variables can be declared implicitly by simply using the variable name within your script. This practice is not recommended. It leads to code that is prone to errors and more difficult to debug.
Also note that VBScript variable names are *not case sensitive*. The variables SumTotal, Sumtotal, and sumtotal, SUMTOTAL all refers to the same variable.

Variable names can be made up of letters, digits, and the underscore character only (no spaces); must begin with a letter; and cannot be the same as VBScript key words such as **For**, **While**, etc.

3.2.2.2 Variable Naming Rules

When naming variables the following rules apply:

- they must begin with an alphabetic character
- they cannot contain embedded periods
- they must be unique within the same scope. There is more on scopes later in this lesson
- they must be no longer than 255 characters.

3.2.3 Variants and Subtypes

As earlier mentioned, VBScript has a single data type called a *variant*. Variants have the ability to store different types of data. The types of data that a variant can store are referred to as *subtypes*. The table below describes the subtypes supported by VBScript.

Subtype	**Description of Uses for Each Subtype**
Byte	Integer numbers between 0 to 255
Boolean	*True* and *False*
Currency	Monetary values
Date	Date and time
Double	Extremely large numbers with decimal points
Empty	The value that a variant holds before being used
Error	An error number
Integer	Large integers between -32,768 and 32,767
Long	Extremely large integers (-2,147,483,648 and 2,147,483,647)
Object	Objects
Null	No valid data
Single	Large numbers with decimal points
String	Character strings

3.2.3.1 Assigning Values

You assign a value to a variable by using the following format:

a. Variable name = value
b. The following examples demonstrate assigning values to variables:
c. Name = "Larry Roof"
d. WorkHour = 50

e. Overtime = True

3.2.3.2 Scope of Variables

The scope of a variable determines where it can be used in your script. A variable's scope is determined by where it is declared. In terms of scope there are two types of variables

Procedure-level variables: variables declared within a procedure. It is referred to as a *procedure-level* variable and can only be used within that procedure.

Script-level variables: If it is declared outside of any procedure, it is a *script-level* variable and can be used throughout the script.

The example below demonstrates both script-level and procedure-level variables.

- <SCRIPT>
- Dim count
- Sub cmdButton_onClick
- Dim Num
- End Sub
- </SCRIPT>

The variable *count* is a script-level variable and can be utilised throughout the script. The variable *Num* exists only within the *cmdButton_onClick* sub-procedure.

3.2.3.3 Constants

VBScript does not provide support for constants, such as you find in other programming languages. This can be manipulated by assigning values to variables that you have defined. For example TAX_RATE in the below script is a constant.

- <SCRIPT>
- Dim TAX_RATE
- TAX_RATE = .06
- Function CalculateTaxes
- CalculateTaxes = CostOfGoods * TAX_RATE
- End Function
- </SCRIPT>

3.2.4 VBScript Array Variable

The VBScript language provides support for arrays. An **array** is a type of variable that can hold a list of single variables that are referenced by their numerical index starting at 0. The index, also called a subscript, is an integer. Because indexing starts at 0 the first element in an array is ArrayElement(0), the second is ArrayElement(1), etc. The index of the last element in an array will always be 1 less than the number of elements in the array and is called the *upper bound*. The *lower bound* is always 0. You declare arrays with a **Dim** statement (whether or not **Option Explicit** is used) indicating the upper bound so that the correct amount of memory can be reserved:

Dim Courses (50)
The statement above creates an array with 51 elements. Why 51? Because VBScript arrays are *zero-based*, meaning that the first array element is indexed 0 and the last is the number specified when declaring the array.

You assign values to the elements of an array just as you would a variable, but with an additional reference (the index) to the element in which it will be stored:

- Courses (2) = “CSC121”
- Courses (4) = “ACC222”

3.2.4.1 Multiple Dimensions Array

Arrays can have multiple dimensions-VBScript supports up to 60. Declaring a two dimensional array for storing 51 states and their capitals could be done as follows:

- Dim StateInfo(50,1)
- To store values into this array you would then reference both dimensions.
- StateInfo(18,0) = "Ikeja"
- StateInfo(18,1) = "Kano"

3.2.4.2 Dynamic Array

These are arrays whose size may need to change as the script is executing. A dynamic array is declared without specifying the number of elements it will contain:

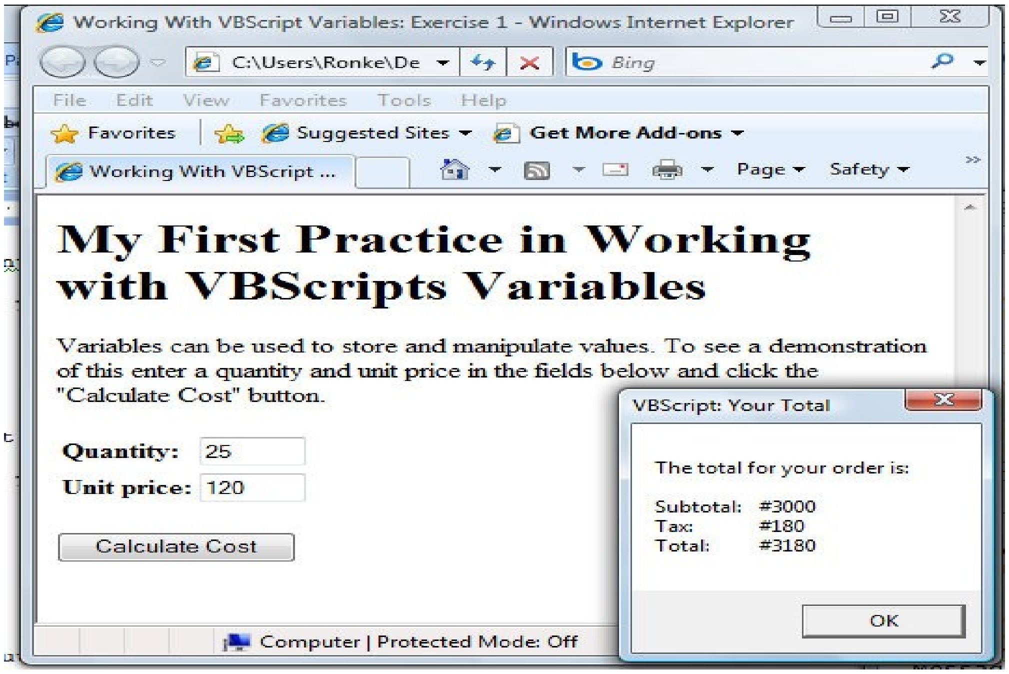
- Dim Courses()

- The *ReDim* statement is then used to change the size of the array from within the script:
- ReDim Courses (100)

There is no limit to the number of times an array can be re-dimensioned during the execution of a script. To preserve the contents of an array when you are re-dimensioning, use the *Preserve* keyword:

- ReDim Preserve Courses (100)
- <HTML>
- <HEAD>
- <TITLE>Working with VBScript Variables: Exercise 1</TITLE>
- <SCRIPT LANGUAGE="VBScript">
- <!-- Add this to instruct non-IE browsers to skip over VBScript modules.
- Option Explicit
- Sub cmdCalculate_OnClick
- Dim AmountofTax
- Dim CRLF
- Dim Message
- Dim Subtotal
- Dim TABSPACE
- Dim TAX_RATE
- Dim TotalCost
- ' Define our constant values.
- TAX_RATE = 0.06
- CRLF = Chr(13) & Chr(10)
- TABSPACE = Chr(9)
- ' Perform order calculations.
- Subtotal = document.frmExercise2.txtQuantity.value _
- document.frmExercise2.txtUnitPrice.value
- AmountofTax = Subtotal * TAX_RATE
- TotalCost = Subtotal + AmountofTax
- ' Display the results.
- Message = "The total for your order is:"
- Message = Message & CRLF & CRLF
- Message = Message & "Subtotal:" & TABSPACE & "#" & Subtotal & CRLF
- Message = Message & "Tax:" & TABSPACE & "#" & AmountofTax & CRLF
- Message = Message & "Total:" & TABSPACE & "#" & TotalCost
- MsgBox Message,,"Your Total"

- End Sub
- -->
- </SCRIPT>
- </HEAD>
- <BODY>
- <H1>My First Practice in Working with VBScripts Variables</H1>
- <P> Variables can be used to store and manipulate values. To
- see a demonstration of this enter a quantity and unit price
- in the fields below and click the "Calculate Cost" button.</P>
- <FORM NAME="frmExercise2">
- <TABLE>
- <TR>
- <TD><B>Quantity:</B></TD>
- <TD><INPUT TYPE="Text" NAME="txtQuantity" SIZE=5></TD>
- </TR>
- <TR>
- <TD><B>Unit price:</B></TD>
- <TD><INPUT TYPE="Text" NAME="txtUnitPrice" SIZE=5></TD>
- </TR>
- </TABLE>
- <BR>
- <INPUT TYPE="Button" NAME="cmdCalculate" VALUE="Calculate Cost">
- </FORM>
- </BODY>
- </HTML>


Working With VBScript Variables: Exercise 1 - Windows Internet Explorer
C:\Users\Ronke\De
Bing
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Get More Add-ons
Working With VBScript ...
Page
Safety
Computer | Protected Mode: Off

My First Practice in Working with VBScripts Variables

Variables can be used to store and manipulate values. To see a demonstration of this enter a quantity and unit price in the fields below and click the "Calculate Cost" button.

Quantity: 25
Unit price: 120

Calculate Cost

VBScript: Your Total

```
The total for your order is:

Subtotal:   #3000
Tax:        #180
Total:      #3180
```

OK

Fig.3.2: Example 2

Brief explanation
After the starting <*SCRIPT*> tag and HTML comment we find:

Option Explicit
This statement forces you to declare all of your variables.
Next we create a sub procedure for the click event of the *cmdCalculate* button.

Sub cmdCalculate_OnClick
Following that seven variables were declared, three of which were used as constants. They were identified by the fact that they are all in uppercase. In VBScript, case doesn't matter (though it does in JavaScript). Note also that the variables are procedure-level since they are declared within a procedure.

In VBScript, anything after an apostrophe is a comment. As such, they are ignored when the script is processed. Comments can appear on a line by themselves or at the end of a line of script. Comments at the end of a line are referred to as inline comments.

' Define our constant values.
The constants are assigned values in the following lines. *Chr()* is a VBScript function that returns the character associated with a specified ASCII code. ASCII codes 13, 10 and 9 are carriage return, line feed and tab, respectively.

- CRLF = Chr(13) & Chr(10)
- TABSPACE = Chr(9)

The next line demonstrates how values are taken from a form on a web page, and used within a script. The two fields on our form were named *txtQuantity* and *txtUnitPrice* in their HTML <*INPUT*> tags. The form was named *frmExercise2*. Here we are referencing our web document, then the form, then the input field and finally the *value* of that field. The value associated with each field contains what the user entered into that field on the web page. The * says to multiply the value of the first field, *txtQuantity*, by the second field, *txtUnitPrice*.

Note
The commonly used VBScript operands are + for addition, - for subtraction, * for multiplication and / for division.

The result of this calculation is then stored in the variable *Subtotal*. Next we perform some additional calculations. Finally, we display the result of our calculations using the *MsgBox* function. The ampersand character, *&*, is used to concatenate the two strings.

As with the previous lesson, don't get too worried about understanding all of the details of this example right now. As you continue to work with VBScript you will begin to "pickup" the language.

SELF-ASSESSMENT EXERCISE

i. What are the benefits web pages achieve by using VBScript.
ii. List the browsers supported by VBScript.
iii. How does VBScript differ from JavaScript and which proves better. State the reason for the same.

4.0 CONCLUSION

At the end of this unit, we have discussed the VBScript Data types, Variables, and VBScript Program Control statements.

5.0 SUMMARY

VBScript is a scripting language which was designed as an extension to HTML. It is a light version of Visual Basic programming language. VBScript also requires a little understanding of HTML.

6.0 TUTOR-MARKED ASSIGNMENT

i. Develop the order processing page for PHONEX ONLINE MOBILE PHONE SHOP in Module Two Unit Two using VBScript. Include a hyperlink called “VIEW STOCK” for customers to access available items.

ii. Develop a page for the link “VIEW STOCK” above using the information below

NAME	MODEL	SELLING PRICE
Nokia	2600	#8,500
Samsung	S510	#12,000
Motorola	M4100	13,000
Nokia	5100c	12,800

7.0 REFERENCES/FURTHER READING

Ed, W. (2006). “Microsoft VBScript: Step by Step”. Microsoft Press.

Don, J. & Jeffery, H. (2006). *Advanced VBScript for Microsoft Windows Administrator*s. Microsoft Press.

UNIT 4 VBSCRIPT OBJECT AND CONTROLLING VBSCRIPT ROUTINES

CONTENTS

1.0 INTRODUCTION

This unit is divided into two parts: i) Objects and VBScript Objects and ii) controlling your VBScript Routines. Objects both in the form of Java applets and ActiveX controls, enhances the functionality that is provided with HTML. By using VBScript objects you can extend the capabilities of controls, integrating and manipulating them from within your scripts.

Controlling Routines: VBScript allows you to control how your scripts process data through the use of *conditional* and *looping* statements. By using conditional statements you can develop scripts that evaluate data and use criteria to determine what tasks to perform.

2.0 OBJECTIVES

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

- outline the use of: Background, Text, Font, Box Model, font, describe how to utilise the power of objects in VBScript
- explain how to add ActiveX controls
- explain how the script process the data through the use of control statements.

3.0 MAIN CONTENT

Adding Objects to your Web Pages

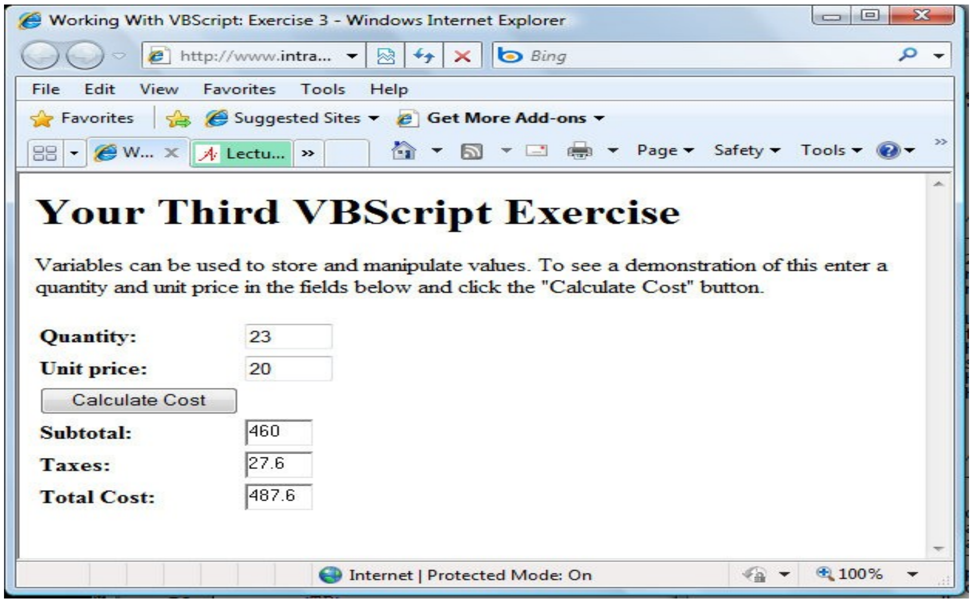
Scripting with objects involves two steps:

Adding the object to your web page using HTML
Writing script procedures to respond to events that the object provides
Objects, whether they're Java applets or ActiveX controls are added to a page with the <*OBJECT*> tag. The properties, or characteristics, of the object are configured using the <*PARAM*> tag. Typically you will see an object implemented using a single <*OBJECT*> tag along with several <*PARAM*> tags. The following HTML code demonstrates how an ActiveX control might appear when added to a page:

- <OBJECT ID="lblTotalPay" WIDTH=45 HEIGHT=24
- CLASSID="CLSID:978C9E23-D4B0-11CE-BF2D-00AA003F40D0">
- <PARAM NAME="ForeColour" VALUE="0">
- <PARAM NAME="BackColour" VALUE="16777215">
- <PARAM NAME="Caption" VALUE="">
- <PARAM NAME="Size" VALUE="1582;635">
- <PARAM NAME="SpecialEffect" VALUE="2">
- <PARAM NAME="FontHeight" VALUE="200">
- <PARAM NAME="FontCharSet" VALUE="0">
- <PARAM NAME="FontPitchAndFamily" VALUE="2">
- <PARAM NAME="FontWeight" VALUE="0">

3.1 Linking VBScript with Objects

Once you have added a control to your web page, it can be configured, manipulated and responded to through its properties, methods and events. *Properties* are the characteristics of an object. They include items like a caption, the foreground colour and the font size. *Methods* cause an object to perform a task. *Events* are actions that are recognised by an object. For instance, a command button recognises an *onclick* event.


Working With VBScript: Exercise 3 - Windows Internet Explorer
File Edit View Favorites Tools Help
Calculate Cost
Internet | Protected Mode: On

Your Third VBScript Exercise

Variables can be used to store and manipulate values. To see a demonstration of this enter a quantity and unit price in the fields below and click the "Calculate Cost" button.

Quantity:	23
Unit price:	20
Subtotal:	460
Taxes:	27.6
Total Cost:	487.6

Fig.4.1: Example 1

Brief Explanation

This Example 1 was just a modification of Example 2 in unit three. It contains three ActiveX label controls named *lblSubtotal*, *lblTaxes* and *lblTotalCost*. There were minimal changes involving variable declarations and the defining of constant values. The way results are displayed is different in Example 2 in unit three. The script has been modified to remove the *MsgBox* function and in its place we set the *caption* property of three label controls.

' Display the results.

a. document.frmExercise3.lblSubtotal.caption = Subtotal
b. document.frmExercise3.lblTaxes.caption = AmountofTax
c. document.frmExercise3.lblTotalCost.caption = TotalCost

The format used when referencing properties is:

Document	Our web document
frmExercise3	The form on which the ActiveX controls were placed
lblTaxes	The name of the control
Caption	The property to set

3.2 Conditional Statement

VBScript provides two forms of conditional statements:

- *If..Then..Else*
- *Select..Case*

3.2.1 If..Then..Else

The *If..Then..Else* statement is used, first to evaluate a condition to see if it is true or false and second, depending upon the condition, to execute a statement or set of statements.

The simplest version of an *if* statements is one that contains only one condition and a single statement. For example:

- **If AmountPurchased > 10000 Then** *'a condition to check if the amount purchased is greater than 10000*
- **DiscountAmount = AmountPurchased * .10** 'statement to be executed if the condition is true.

A more complicated version of the *If* statement executes a series of statements when the condition is true, for example:

- If AmountPurchased > 10000 Then
- DiscountAmount = AmountPurchased * .10
- Subtotal = AmountPurchased - DiscountAmount
- End If

In this form of the *If* statement, one or more statements can be executed when the condition is true, by placing them between the *If* statement on top and the *End If* statement on the bottom.

The next form of the *If* statement uses the *If..Then..Else* format. This version of the *If* statement differs from the two previous versions in that it will perform one set of statements if the condition is true and another set when the condition is false:

- If AmountPurchased > 10000 Then
- DiscountAmount = AmountPurchased * .10
- Subtotal = AmountPurchased - DiscountAmount
- Else
- HandlingFee = AmountPurchased *.03
- Subtotal = AmountPurchased + HandlingFee
- End If

3.2.2 Select Case

The *Select Case* statement provides an alternative to the *If..Then..Else* statement, providing additional control and readability when evaluating complex conditions. It is well suited for situations where there are a number of possible conditions for the value being checked. Like the *If* statement the *Select Case* structure checks a condition, and based upon that condition being true, executes a series of statements.

The syntax of the *Select Case* statement is:

- Select Case condition
- Case value
- Case value
- ...
- Case Else
- End Select

For example, the following *Select* statement assigns different lecture fees based upon the course unit where the order is being sent:

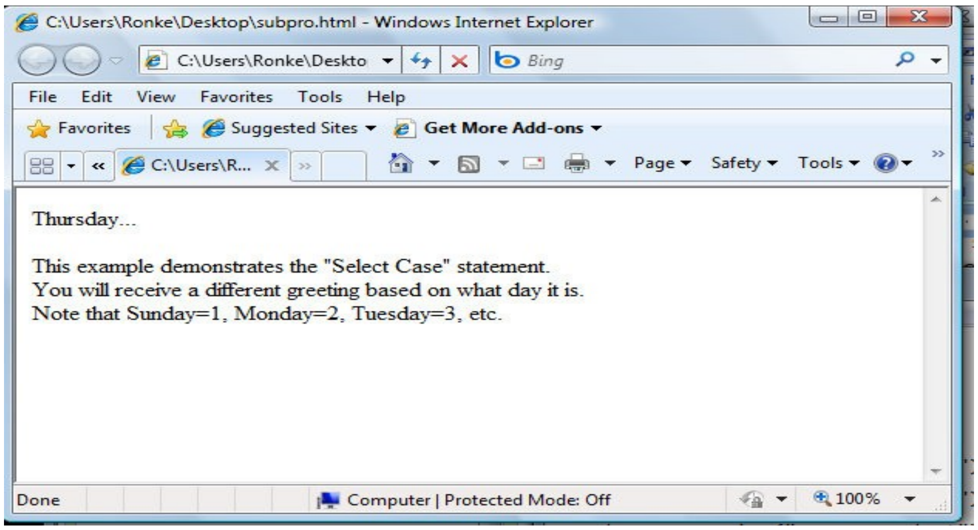
- Select Case Document.frmOrder.txtState.Value
- Case "1-Unit"
- LectureFee= 500
- Case "2-Unit"
- LectureFee = 1,000
- Case Else
- LectureFee = 1,500
- End Select

The *Select Case* statement checks each of the *Case* statements until it finds one that will result in the condition being true. If none are found to be true, it executes the statements within the *Case Else*.

Note
Even though it is not required, always include a *Case Else* when working with *Select Case* statements to process conditions that you may not have considered possible.

Example 2:
- <html>
- <body>
- <script type="text/VBScript">
- d=weekday(Date)

- Select Case d
- Case 1
- document.write("Sleepy Sunday")
- Case 2
- document.write("Monday again!")
- Case 3
- document.write("Just Tuesday!")
- Case 4
- document.write("Wednesday!")
- Case 5
- document.write("Thursday...")
- Case 6
- document.write("Finally Friday!")
- Case Else
- document.write("Super Saturday!!!!")
- End Select
- </script>
- <p>This example demonstrates the "Select Case" statement.<br />
- You will receive a different greeting based on what day it is.<br />
- Note that Sunday=1, Monday=2, Tuesday=3, etc.</p>
- </body>
- </html>


C:\Users\Ronke\Desktop\subpro.html - Windows Internet Explorer
File Edit View Favorites Tools Help
Done
Computer | Protected Mode: Off
100%

```
Thursday...

This example demonstrates the "Select Case" statement.
You will receive a different greeting based on what day it is.
Note that Sunday=1, Monday=2, Tuesday=3, etc.
```

Fig.4.2: Output of Example 2

SELF-ASSESSMENT EXERCISE

i. Distinguish between the followings: property, method, and events.
ii. Develop a simple web page to output the model and prices of phones available when a particular name (e.g. Nokia) is enter. (HINT: use SELECT CASE Statement).

3.3 Controlling Your VBScript Routines

3.3.1 Looping Statement

VBScript provides three forms of looping statements:

- *For..Next*
- *For Each..Next*
- *Do..Loop*

These four statements can be divided into two groups. The *For* statements (For..Next and For Each..Next) are best used when you want to perform a loop a specific number of times. The *Do..While* statements is best used to perform a loop an undetermined number of times.

3.3.1.1 For..Next

The *For..Next* structure is used when you want to perform a loop a specific number of times. It uses a counter variable, which is incremented or decremented with each repetition of the loop. Example:

- For counter = 1 To 12
- result = 5 * counter
- MsgBox counter & " times 5 is " & result
- Next counter

In this example, the variable *counter* is the numeric value being incremented or decremented. The number 1, defines the start of the loop, 12 the end of the loop. When this loop executes it will display twelve dialog box messages, each containing the product of multiplying five times the counter as it runs from 1 to 12.

You can also control how you want your counter to be modified through the addition of the *Step* argument. Example:

- For counter = 1 To 12 Step 2
- result = 5 * counter

- MsgBox counter & " times 5 is " & result
- Next counter

This little modification to the loop results in only the products of the odd numbers between 1 and 12 being displayed because the loop will execute by increasing the counter by 2 at each execution.

3.3.1.2 For Each..Next

The *For Each..Next* is similar to the *For..Next* loop but instead of repeating a loop for a certain number of times, it repeats the loop for each member of a specified collection.

3.3.1.3 Do...Loop

If you don't know how many repetitions you want, use a Do...Loop statement.

The Do...Loop statement repeats a block of code while a condition is true, or until a condition becomes true.

There are three variant of Do..Loop:

- ***Do..While***

You use the While keyword to check a condition in a Do...Loop statement.

Format:

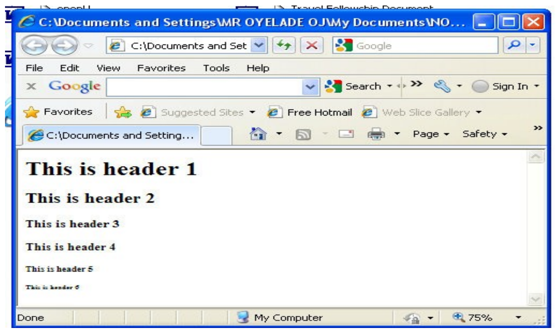
- Do While i>10
- some code
- Loop
- ***Do..Until***
- Format:
- Do
- some code
- Loop Until i=10
- or
- Do Until i=10
- i=i-1
- If i<10 Then Exit Do
- Loop

Example 3

Develop a simple web page using VBScript to loop through the six headings in HTML.

- Solution
- <html>
- <body>
- <script type="text/VBScript">
- For i=1 To 6
- document.write("<h" & i & ">This is header " & i & "</h" & i & ">")
- Next
- </script>
- </body>
- </html>

The output is displayed below:


C:\Documents and Settings\MR OYELADE OJ\My Documents\NO...
File Edit View Favorites Tools Help
Google Search Sign In
Favorites Suggested Sites Free Hotmail Web Slice Gallery
C:\Documents and Setting...
Page Safety
Done My Computer 75%

This is header 1

This is header 2

This is header 3

This is header 4

This is header 5

This is header 6

Fig.4.3: Output of Example 3

SELF-ASSESSMENT EXERCISE

In **Activity A** above, add functionality that allows Customers to supply quantity as input and output the total price in return.

4.0 CONCLUSION

In this unit, we discussed: i) Objects and VBScript Objects and ii) Controlling your VBScript Routines which enhances the functionality that is provided with HTML. By using VBScript objects you can extend the capabilities of controls, integrating and manipulating them from within your scripts.

5.0 SUMMARY

In this unit, we have learnt:

- how to utilise the power of objects in VBScript
- how to add ActiveX controls
- how the script process the data through the use of control statements.

6.0 TUTOR-MARKED ASSIGNMENT

Modify your web page in Unit 3, Tutor- marked assignment 2 to accept: phone model and quantity as input and consequently compute and display: subtotal, taxes, and total cost.

7.0 REFERENCES/FURTHER READING

Ed, Wilson (2006). *Microsoft VBScript*: *Step by Step*. Microsoft Press.

Don, J. & Jeffery, H. (2006).*Advanced VBScript for Microsoft Windows* Administrators. Microsoft Press.

MODULE 4 EXTENSIBLE MARKUP LANGUAGE (XML)

UNIT 1 XML BASICS

CONTENTS

1.0 INTRODUCTION

XML stands for Extensible Mark-up Language. A mark-up language specifies the structure and content of a document. Because it is extensible, XML can be used to create a wide variety of document types. XML (eXtensible Mark-up Language) is a meta-language; that is, it is a language in which other languages are created. In XML, data is "marked up" with tags, similar to HTML tags. XML is used to store data or information. This unit will introduce you to XML basics, XML benefits.

2.0 OBJECTIVES

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

- define XML
- explain the benefits XML
- list XML syntax rules
- describe how to create a simple XML file.

3.0 MAIN CONTENT

Definition of XML
XML stands for Extensible Mark-up Language. A mark-up language specifies the structure and content of a document. XML is a subset of the Standard Generalised Mark-up Language (SGML) which was introduced in the 1980s. SGML is very complex and can be costly. These reasons led to the creation of Hypertext Mark-up Language (HTML), a more easily used mark-up language. XML can be seen as sitting between SGML and HTML – easier to learn than SGML, but more robust than HTML.

3.1 The 10 Primary XML Design Goals

1. XML must be easily usable over the Internet
2. XML must support a wide variety of applications
3. XML must be compatible with SGML
4. It must be easy to write programs that process XML documents
5. The number of optional features in XML must be kept small
6. XML documents should be clear and easily understood
7. The XML design should be prepared quickly
8. The design of XML must be exact and concise
9. XML documents must be easy to create
10. Keeping an XML document size small is of minimal importance.

3.2 Benefits of XML

Initially, XML received a lot of excitement, which has now died down.

This is not because XML is not as useful, but rather because it does not provide instant attractive factor that others technology, such as HTML do. When you write an HTML document, you see a nicely formatted page in a browser - instant gratification. When you write an XML document, you see an XML document - not so exciting. However, with a little more effort, you can make that XML document more robust.

This section discusses some of the major benefits of XML.

3.2.1 XML Holds Data

XML does not really do much of anything. Rather, developers can create XML-based languages that store data in a structure way. Applications can then use this data to do any number of things.

3.2.2 XML Separates Structure from Formatting

One of the difficulties with HTML documents, Word Processor documents, Spreadsheets, and other forms of documents is that, they mix structure with formatting. This makes it difficult to manage content and design, because the two are intermingled.

As an example, in HTML, there is a <u> tag used for underlining text. Very often, it is used for emphasis, but it also might be used to mark a book title. It would be very difficult to write an application that searched through such a document for book titles.

In XML, the book titles could be placed in <book title> tags and the emphasied text could be place in <em> tags. The XML document does not specify how the content of either tag should be displayed. Rather, the formatting is left up to an external styleSheet. Even though the book titles and emphasised text might appear the same, it would be relatively straight forward to write an application that finds all the book titles. It would simply look for text in <book_title> tags. It also becomes much easier to reformat a document; for example, to change all emphasised text to be italicised rather than underlined, but leave book titles underlined.

3.2.3 XML Promotes Data Sharing

Very often, applications that hold data in different structures must share data with one another. It can be very difficult for a developer to map the different data structures to each other. XML can serve as a go between. Each application's data structure is mapped to an agreed-upon XML structure. Then all the applications share data in this XML format. Each application only has to know two structures, its own and the XML structure, to be able to share data with many other applications.

3.2.4 XML is Human-Readable

XML documents are (or can be) read by people. Perhaps this doesn't sound so exciting, but compare it to data stored in a database. It is not easy to browse through a database and read different segments of it as you would a text file. Take a look at the XML document below.

- Code Sample: oyelade.xml
- <?xml version="1.0"?>
- <person>
- <name>
- <firstname>Olanrewaju</firstname>
- <lastname>Oyelade</lastname>
- </name>
- <job>Lecturer</job>
- <gender>Male</gender>
- </person>

Code explanation
It is not hard to tell from looking at this that the XML is describing a person named Olanrewaju Oyelade, who is a Lecturer and is a male.

3.2.5 XML is Free

XML doesn't cost anything to use. It can be written with a simple text editor or one of the many freely available XML authoring tools, such as XML Notepad. In addition, many web development tools, such as Dreamweaver and Visual Studio .NET have built-in XML support. There are also many free XML parsers, such as Microsoft's MSXML (downloadable from microsoft.com) and Xerces (downloadable at apache.org).

3.3 XML in Practice

3.3.1 Content Management

Almost all of the leading content management systems use XML in one way or another. A typical use would be to store a company's marketing content in one or more XML documents. These XML documents could then be transformed for output on the Web, as Word documents, as PowerPoint slides, in plain text, and even in audio format. The content can also easily be shared with partners who can then output the content in their own formats.

Storing the content in XML makes it much easier to manage content for two reasons.

Content changes, additions, and deletions are made in a central location and the changes will cascade out to all formats of presentation. There is no need to be concerned about keeping the Word documents in sync with the website, because the content itself is managed in one place and then transformed for each output medium.

Formatting changes are made in a central location. To illustrate, suppose a company had many marketing web pages, all of which were produced from XML content being transformed to HTML. The format for all of these pages could be controlled from a single XSLT and a sitewide formatting change could be made modifying that XSLT.

3.3.2 Web Services

XML Web services are small applications or pieces of applications that are made accessible on the Internet using open standards based on XML. Web services generally consist of three components:

- SOAP - an XML-based protocol used to transfer Web services over the Internet.
- WSDL (Web Services Description Language) - an XML-based language for describing a Web service and how to call it.
- Universal Discovery Description and Integration (UDDI) - the yellow pages of Web services. UDDI directory entries are XML documents that describe the Web services a group offers. This is how people find available Web services.

3.3.3 RDF / RSS Feeds

RDF (Resource Description Framework) is a framework for writing XML-based languages to describe information on the Web (e.g. web pages). RSS (RDF Site Summary) is an implementation of this framework; it is a language that adheres to RDF and is used to describe web content. Website publishers can use RSS to make content available as a "feed", so that web users can access some of their content without actually visiting their site. Often, RSS is used to provide summaries with links to the company's website for additional information.

3.4 XML Documents

An XML document is made up of the following parts:

a. An optional prolog
b. A document element, usually containing nested elements
c. Optional comments or processing instructions.

3.4.1 The XML Declaration

The XML declaration, if it appears at all, must appear on the very first line of the document with no preceding white space. It looks like this.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

This declares that the document is an XML document. The version attribute is required, but the encoding and standalone attributes are not. If the XML document uses any markup declarations that set defaults for attributes or declare entities then standalone must be set to no.

3.4.2 Processing Instructions

Processing instructions are used to pass parameters to an application. These parameters tell the application how to process the XML document. For example, the following processing instruction tells the application that it should transform the XML document using the XSL stylesheet oyelade.xsl.

```
<?xml-stylesheet href="oyelade.xsl" type="text/xsl"?>
```
As shown above, processing instructions begin with <? and end with ?>.

3.4.3 Comments

Comments can appear throughout an XML document. Like in HTML, they begin with <!-- and end with -->.
<!--This is a comment-->

3.4.4 A Document Type Declaration

The Document Type Declaration (or DOCTYPE Declaration) has three roles:

- it specifies the name of the document element
- it may point to an external Document Type Definition (DTD)
- it may contain an internal DTD.

The DOCTYPE Declaration shown below simply states that the document element of the XML document is beatles.
<!DOCTYPE beatles>

If a DOCTYPE Declaration points to an external DTD, it must either specify that the DTD is on the same system as the XML document itself or that it is in some public location. To do so, it uses the keywords SYSTEM and PUBLIC. It then points to the location of the DTD using a relative Uniform Resource Indicator (URI) or an absolute URI. Here are a couple of examples.

Syntax
- <!--DTD is on the same system as the XML document-->
- <!DOCTYPE beatles SYSTEM "dtds/oyelade.dtd">

Syntax
- <!--DTD is publicly available-->
- <!DOCTYPE oyelade PUBLIC "-//Openuniversity//DTD Oyelade 1.0//EN"
- "http://www.openuniversity.com/oyelade/DTD/oyelade.dtd">

As shown in the second declaration above, public identifiers are divided into three parts:

a. An organisation (e.g, Openuniversity)
b. A name for the DTD (e.g, Oyelade 1.0)
c. A language (e.g. EN for English).

3.5 Elements

Every XML document must have at least one element, called the document element. The document element usually contains other elements, which contain other elements, and so on. Elements are denoted with tags. Let's look again at the Oyelade.xml.

- Code Sample: Oyelade.xml
- <?xml version="1.0"?>
- <person>
- <name>
- <firstname>Olanrewaju</firstname>
- <lastname>Oyelade</lastname>
- </name>
- <job>Lecturer</job>
- <gender>Male</gender>
- </person>

Code Explanation
The document element is person. It contains three elements: name, job and gender. Further, the name element contains two elements of its own: firstname and lastname. As you can see, XML elements are denoted with tags, just as in HTML. Elements that are nested within another element are said to be children of that element.

3.5.1 Empty Elements

Not all elements contain other elements or text. For example, in XHTML, there is an img element that is used to display an image. It does not contain any text or elements within it, so it is called an empty element. In XML, empty elements must be closed, but they do not require a separate close tag. Instead, they can be closed with a forward slash at the end of the open tag as shown below.

```
<img src="images/oyelade.jpg"/>
```

The above code is identical in function to the code below.

```
<img src="images/oyelade.jpg"></img>
```

SELF-ASSESSMENT EXERCISE

i. What is the difference(s) between XML and HTML.
ii. State ten Primary goals of XML Design.
iii. What are the benefits of XML over HTML.

iv. Write short notes on the following:
    a. SOAP
    b. UDDI and
    c. WSDL

3.6 Attributes

XML elements can be further defined with attributes, which appear inside of the element's open tag as shown below.

- *Syntax*
- <name title="Sir">
- <firstname>Olanrewaju</firstname>
- <lastname>Oyelade</lastname>
- </name>

3.7 CDATA

Sometimes it is necessary to include sections in an XML document that should not be parsed by the XML parser. These sections might contain content that will confuse the XML parser, perhaps because it contains content that appears to be XML, but is not meant to be interpreted as XML. Such content must be nested in CDATA sections. The syntax for CDATA sections is shown below.

Syntax

```
<![CDATA[

This section will not get parsed

 by the XML parser.
]]>
```

3.8 White Space

In XML data, there are only four white space characters.

- Tab
- Line-feed
- Carriage-return
- Single space
- There are several important rules to remember with regards to white space in XML.

- White space within the content of an element is significant; that is, the XML processor will pass these characters to the application or user agent.
- White space in attributes is normalised; that is, neighbouring white spaces are condensed to a single space.
- White space in between elements is ignored.

3.8.1 xml:space Attribute

The xml:space attribute is a special attribute in XML. It can only take one of two values: default and preserve. This attribute instructs the application how to treat white space within the content of the element. Note that the application is not required to respect this instruction.

3.9 XML Syntax Rules

XML has relatively straightforward, but very strict, syntax rules. A document that follows these syntax rules is said to be well-formed.

There must be one and only one document element.

Every open tag must be closed.

If an element is empty, it still must be closed.

Poorly-formed: <tag>
Well-formed: <tag></tag>
Also well-formed: <tag />

Elements must be properly nested.

Poorly-formed: <a><b></a></b>
Well-formed: <a><b></b></a>

Tag and attribute names are case sensitive.
Attribute values must be enclosed in single or double quotes.

3.9.1 Special Characters

There are five special characters that can not be included in XML documents. These characters are replaced with predefined entity references as shown in the table below.

Special Characters	
Character	**Entity Reference**
<	&lt;
>	&gt;
&	&amp;
"	&quot;
'	&apos;

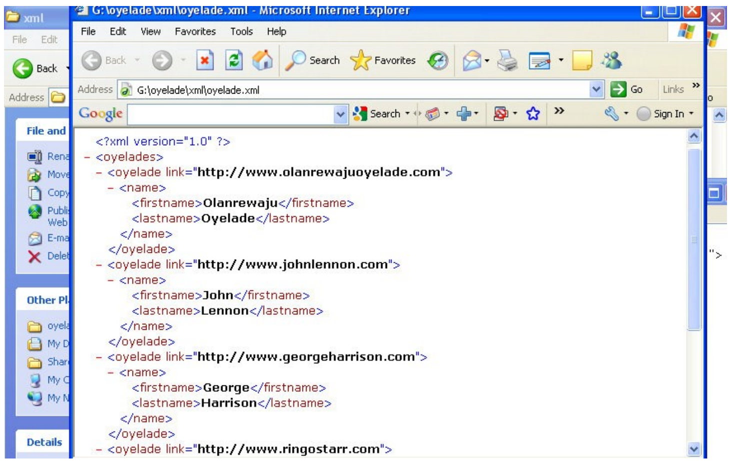
3.10 Creating a Simple XML File

The following is relatively simple XML file describing the Oyelades.

- Code Sample: Oyelade.xml
- <?xml version="1.0"?>
- <oyelades>
- <oyelade link="http://www.olanrewajuoyelade.com">
- <name>
- <firstname>Olanrewaju</firstname>
- <lastname>Oyelade</lastname>
- </name>
- </oyelade >
- <oyelade link="http://www.johnlennon.com">
- <name>
- <firstname>John</firstname>
- <lastname>Lennon</lastname>
- </name>
- </oyelade >
- <oyelade link="http://www.georgeharrison.com">
- <name>
- <firstname>George</firstname>
- <lastname>Harrison</lastname>
- </name>
- </oyelade>
- <oyelade link="http://www.ringostarr.com">
- <name>
- <firstname>Ringo</firstname>
- <lastname>Starr</lastname>
- </name>
- </oyelade>
- <oyelade link="http://www.openuniversity.com" real="no">
- <name>
- <firstname>Dada</firstname>

- <lastname>Ojo</lastname>
- </name>
- </oyelade>
- </oyelades>

The output of Oyelade.xml is shown below:


G:\oyelade\xml\oyelade.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address G:\oyelade\xml\oyelade.xml
Go Links
Google Search Sign In

```
<?xml version="1.0" ?>
- <oyelades>
  - <oyelade link="http://www.olanrewajuoyelade.com">
    - <name>
        <firstname>Olanrewaju</firstname>
        <lastname>Oyelade</lastname>
      </name>
    </oyelade>
  - <oyelade link="http://www.johnlennon.com">
    - <name>
        <firstname>John</firstname>
        <lastname>Lennon</lastname>
      </name>
    </oyelade>
  - <oyelade link="http://www.georgeharrison.com">
    - <name>
        <firstname>George</firstname>
        <lastname>Harrison</lastname>
      </name>
    </oyelade>
  - <oyelade link="http://www.ringostarr.com">
```

SELF-ASSESSMENT EXERCISE

i. In this Activity, you will be modifying an existing XML file.
ii. Open Xml101.xml
iii. Add a required prerequisite: "Experience with computers".

Add the following to the topics list:

- XML Documents
- The Prolog Elements
- Attributes
- CDATA
- XML Syntax Rules
- Special Characters
- Creating a Simple XML File
- Add a modifications element that shows the modifications you've made.

Note: The required file is given below

- *Code Sample: Xml101.xml*
- <?xml version="1.0"?>
- <course>
- <head>

- <title>Internet Concept and Web Design</title>
- <course_num>CIT853</course_num>
- <course_unit>3 </course_unit>
- </head>
- <body>
- <prerequisites>
- <prereq>Experience with Word Processing</prereq>
- <prereq optional="true">Experience with HTML</prereq>
- **<!--** Add a required prerequisite: "Experience with computers" -->
- </prerequisites>
- <outline>
- <topics>
- <topic>XML Basics
- <topics>
- <topic>What is XML?</topic>
- <topic>XML Benefits
- <topics>
- <topic>XML Holds Data, Nothing More</topic>
- <topic>XML Separates Structure from Formatting</topic>
- <topic>XML Promotes Data Sharing</topic>
- <topic>XML is Human-Readable</topic>
- <topic>XML is Free</topic>
- </topics>
- </topic>
- **<!--**

Add the following to the topics list ("XML Documents" and "Creating a Simple XML File" should be siblings of "What is XML?" and "XML Benefits"):

- XML Documents
- The Prolog
- Elements
- Attributes
- CDATA
- XML Syntax Rules
- Special Characters
- Creating a Simple XML File

-->
</topics>
</topic>
</topics>
</outline>

```
</body>
<foot>
 <creator>Olanrewju Oyelade</creator>
<date_created>2009-08-25</date_created>
<modifications madeby="George Germond" date="2009-09-05">
<modification type="insert">Added HTML prerequisite</modification>
<modification type="edit">Fixed some typos</modification>
 </modifications>
 <!--
 Add a modifications element that shows the modifications you've made.
 -->
 </foot>
</course>
```

4.0 CONCLUSION

XML stands for Extensible Mark-up Language which can be used to create a wide variety of document types. XML (eXtensible Mark-up Language) is a meta-language; and it is used to store data or information.

5.0 SUMMARY

In this unit, we have learnt:

- the benefits of XML
- how XML is used, and
- how to write a well-formed XML document.

6.0 TUTOR-MARKED ASSIGNMENT

Edit and modify the XML document below using any text editor, display it on a web browser and debug the file with the help of a browser.

Hits.txt

```
<?XML VERSION="1.0" encoding="UTF-8" standalone="yes" ?>
<!-- This file has some errors.  Find and fix errors. Add your name and
the current date here -->
<CD>
<CDTITLE>All time greatest hits <!--Live music </CDTITLE> -->
<ARTIST>Louis Armstrong</artist>
        <track>What a wonderful world</track>
        <track>Hello Dolly</track>
        <track>Kiss in fire</track>
        <track>Mack the knife</track>
</CD>
```

To complete this task, the students should do the following steps:

i. Using any text editor, create the Hits.txt document and save the document as Hits.xml.
ii. Within the document's prolog, insert the name and the date.
iii. Check the syntax of elements and comments. Fix errors and save the document with the same name.
iv. Open the Hits.xml in a Web browser. If a syntax error is found, an error message will be displayed instead of the document's contents.
v. Open Hits.xml in the text editor and fix the error. Save the document with the same name. Repeat steps 4 and 5 until the document's content are displayed.
vi. Check the content and find any logical errors or missing data. Fix logical errors. Save the document with the same name.
vii. Print correct version of Hits.xml and the web page generated by the browser.

7.0 REFERENCES/FURTHER READING

Brooks & David, R. (2007). An Introduction to HTML and JavaScript for Scientists and Engineers.

David, F. (1998). *JavaScript*: *The Definitive Guide*. (3rd ed.). O'Reilly Media.

Larry, R. (2002).“Programming the Web Using XHTML and JavaScript”, McGraw-Hill.

UNIT 2 DOCUMENT TYPE DEFINITION (DTDS)

CONTENTS

1.0 INTRODUCTION

This unit introduces the use of Document Type Definitions (DTDs) which is used to ensure that all required elements are present in the document and also to prevent undefined elements from being used and enforce a specific data structure. It also specifies the use of attributes and defines their possible values and also defines default values for attributes. DTD also describe how the parser should access non-XML or non-textual content.

2.0 OBJECTIVES

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

- state the difference between well-formed and valid XML documents.
- describe the purpose of DTDs
- explain how to create internal and external DTDs
- explain how to validate an XML document according to a DTD.
- enumerate the limitations of DTDs.

3.0 MAIN CONTENT

Well-formed vs. Valid
A well-formed XML document is one that follows the syntax rules described in "XML Syntax Rules". A valid XML document is one that conforms to a specified structure. For an XML document to be validated, it must be checked against a schema, which is a document that defines the structure for a class of XML documents. XML documents that are not intended to conform to a schema can be well-formed, but they cannot be valid.

3.1 The Purpose of DTDs

A Document Type Definition (DTD) is a type of schema. The purpose of DTDs is to provide a framework for validating XML documents. By defining a structure that XML documents must conform to, DTDs allow different organisations to create shareable data files.

Imagine, for example, a company that creates technical courseware and sells it to technical training companies. Those companies may want to display the outlines for that courseware on their websites, but they do not want to display it in the same way as every other company who buys the courseware. By providing the course outlines in a predefined XML format, the courseware vendor makes it possible for the training companies to write programs to read those XML files and transform them into HTML pages with their own formatting styles (perhaps using XSLT or CSS). If the XML files had no predefined structure, it would be very difficult to write such programs.

3.2 Creating DTDs

DTDs are simple text files that can be created with any basic text editor. Although they look a little cryptic at first, they are not terribly complicated once you get used to them.

A DTD outlines what elements can be in an XML document and the attributes and subelements that they can take. Let's start by taking a look at a complete DTD and then dissecting it.

- Code Sample: DTDs/Demos/Oyelade.dtd
- ***<!ELEMENT oyelades (oyelade+)>***
- ***<!ELEMENT oyelade (name)>***
- ***<!ATTLIST oyelade***
- ***link CDATA #IMPLIED***
- ***real (yes|no) "yes">***
- ***<!ELEMENT name (firstname, lastname)>***

- ***<!ELEMENT firstname (#PCDATA)>***
- ***<!ELEMENT lastname (#PCDATA)>***

3.2.1 The Document Element

When creating a DTD, the first step is to define the document element.

<!ELEMENT oyelades (oyelade+)>

The element declaration above states that the oyelades element must contain one or more oyelade elements.

3.2.2 Child Elements

When defining child elements in DTDs, you can specify how many times those elements can appear by adding a modifier after the element name. If no modifier is added, the element must appear once and only once. The other options are shown in the table below.

Modifier	**Description**
?	Zero or one times.
+	One or more times.
*	Zero or more times.

It is not possible to specify a range of times that an element may appear (e.g. 2-4 appearances).

3.2.3 Other Elements

The other elements are declared in the same way as the document element - with the <!ELEMENT> declaration. The Oyelade DTD declares four additional elements.

Each oyelade element must contain a child element name, which must appear once and only once.

<!ELEMENT oyelade (name)>

Each name element must contain a firstname and lastname element, which each must appear once and only once and in that order.

<!ELEMENT name (firstname, lastname)>

Some elements contain only text. This is declared in a DTD as #PCDATA. PCDATA stands for parsed character data, meaning that the data will be parsed for XML tags and entities. The firstname and lastname elements contain only text.

```
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

3.2.4 Choice of Elements

It is also possible to indicate that one of several elements may appear as a child element. For example, the declaration below indicates that an img element may have a child element name or a child element id, but not both.

```
<!ELEMENT img (name|id)>
```

3.2.5 Empty Elements

Empty elements are declared as follows.

```
<!ELEMENT img EMPTY>
```

3.2.6 Mixed Content

Sometimes elements can have elements and text intermingled. For example, the following declaration is for a body element that may contain text in addition to any number of link and img elements.

```
<!ELEMENT body (#PCDATA | link | img)*>
```

3.2.7 Location of Modifier

The location of modifiers in a declaration is important. If the modifier is outside of a set of parentheses, it applies to the group; whereas, if the modifier is immediately next to an element name, it applies only to that element. The following examples illustrate.

In the example below, the body element can have any number of interspersed child link and img elements.

```
<!ELEMENT body (link | img)*>
```

In the example below, the body element can have any number of child link elements or any number of child img elements, but it cannot have both link and img elements.

```
<!ELEMENT body (link* | img*)>
```

In the example below, the body element can have any number of child link and img elements, but they must come in pairs, with the link element preceding the img element.

```
<!ELEMENT body (link, img)*>
```

In the example below, the body element can have any number of child link elements followed by any number of child img elements.

```
<!ELEMENT body (link*, img*)>
```

SELF-ASSESSMENT EXERCISE

i. Write short notes on the followings:
   a. DOCTYPE declaration in XML document
   b. #PCDATA
   c. Well-formed XML document
   d. Valid XML document

3.2.8 Using Parentheses for Complex Declarations

Element declarations can be more complex than the examples above. For example, you can specify that a person element either contains a single name element or a firstname and lastname element. To group elements, wrap them in parentheses as shown below.

```
<!ELEMENT person (name | (firstname,lastname))>
```

3.2.8.1 Declaring Attributes

Attributes are declared using the <!ATTLIST > declaration. The syntax is shown below.

- <!ATTLIST ElementName
- AttributeName AttributeType State DefaultValue?
- AttributeName AttributeType State DefaultValue?>
- ElementName is the name of the element taking the attributes.
- AttributeName is the name of the attribute.
- AttributeType is the type of data that the attribute value may hold. Although there are many types, the most common are CDATA (unparsed character data) and ID (a unique identifier). A list of options can also be given for the attribute type.
- DefaultValue is the value of the attribute if it is not included in the element.
- State can be one of three values: #REQUIRED, #FIXED (set value), and #IMPLIED (optional).

The oyelade element has two possible attributes: link, which is optional and may contain any valid XML text, and real, which defaults to yes if it is not included.

- <!ATTLIST oyelade
- link CDATA #IMPLIED
- real (yes|no) "yes">

3.3 Validating an XML Document with a DTD

The DOCTYPE declaration in an XML document specifies the DTD to which it should conform. In the code sample below, the DOCTYPE declaration indicates the file should be validated against Oyelade.dtd in the same directory.

- Code Sample: DTDs/Oyelade.xml
- <?xml version="1.0"?>
- <!DOCTYPE beatles SYSTEM "Oyelade.dtd">
- <oyelades>
- <oyelade link="http://www.paulmccartney.com">
- <name>
- <firstname>Paul</firstname>
- <lastname>McCartney</lastname>
- </name>
- </oyelade>
- <oyelade link="http://www.johnlennon.com">
- <name>
- <firstname>John</firstname>
- <lastname>Lennon</lastname>
- </name>
- </oyelade>
- <oyelade link="http://www.georgeharrison.com">
- <name>
- <firstname>George</firstname>
- <lastname>Harrison</lastname>
- </name>
- </oyelade>
- <oyelade link="http://www.webucator.com" real="no">
- <name>
- <firstname>Nat</firstname>
- <lastname>Dunn</lastname>
- </name>
- </oyelade>
- </oyelades>

SELF-ASSESSMENT EXERCISE

In this activity, you will write a DTD for the business letter shown below. You will then give your DTD to another student, who will mark up the business letter as a valid XML document according to your DTD. Likewise, you will markup the business letter according to someone else's DTD. Make sure that the XML file contains a DOCTYPE declaration.

Code Sample : BusinessLetter.txt
November 29, 2009

Johnson Olanrewaju
National Open University,
291 Broadway Ave.
Victoria Island, Lagos
Nigeria.

Dear Dr. Olanrewaju:

Along with this letter, I have enclosed the following items:

- two original, execution copies of the Webucator Master Services Agreement
- two original, execution copies of the Webucator Premier Support for
- Developers Services Description between
- Lockwood & Lockwood and Webucator, Inc.

Please sign and return all four original, execution copies to me at your earliest convenience. Upon receipt of the executed copies, we will immediately return a fully executed, original copy of both agreements to you.

Please send all four original, execution copies to my attention as follows:

- Webucator, Inc.
- 4933 Allen Aven. Rd.
- Jamesville, Lagos, Nigeria
- Attn: Bill Smith

If you have any questions, feel free to call me at 08035755778
or e-mail me at bsmith@webucator.com.

4.0 CONCLUSION

A DTD can be used to:

- ensure all required elements are present in the document
- prevent undefined elements from being used
- enforce a specific data structure
- specify the use of attributes and define their possible values
- define default values for attributes
- describe how the parser should access non-XML or non-textual content.

5.0 SUMMARY

In this unit, you have learned:

- the difference between well-formed and valid XML documents
- the purpose of DTDs
- how to create internal and external DTDs
- how to validate an XML document according to a DTD
- the limitations of DTDs.

6.0 TUTOR-MARKED ASSIGNMENT

Oyelade has created another XML document Authors.xml. This document contains a list of authors. For each author he has recorded the author name, the author ID, and the phone number. He forgot to ask Samantha Tracey his phone number. For each author he has entered the information about books. For each published book, he recorded the book ID, the book name, the publisher, and the book version number. Some books do not have a version number. Each author and each book should have a unique ID. He asked you to create an internal DTD that declares all of the elements and attributes in the document. You should also use XML Spy to validate your document against the rules you set up in the DTD.

Authors.xml

```
<?xml version="1.0" ?>
<!-- document type declaration follows -->

<AUTHORS>

 <AUTHOR AUTHORID="A001">
   <AUTHOR_Name Title="Mr.">
```

```
        <First>Ivor</First>
        <Last>Horton</Last>
   </AUTHOR_Name>
   <PHONE>212-443-8875</PHONE>

   <books>
        <book bookID="B0001" version="1.1">
          <book_name>Java Programming for beginners</book_name>
          <Publisher>Course Technology</Publisher>
        </book>
        <book bookID="B0002" version="1.2">
          <book_name>Java Programming for
professionals</book_name>
          <Publisher> Course Technology </Publisher>
        </book>
   </books>
 </AUTHOR>

 <AUTHOR AUTHORID="A002">
   <AUTHOR_Name Title="Ms.">
        <First>Samantha</First>
        <Last>Tracey</Last>
   </AUTHOR_Name>

   <books>
        <book bookID="B0028">
          <book_name>XML and DTD</book_name>
          <Publisher> Course Technology.</Publisher>
        </book>
        <book bookID="B0029" version="1.1">
          <book_name>XML and Schemas</book_name>
          <Publisher> Course Technology </Publisher>
        </book>
   </books>
 </AUTHOR>
```

To complete this task, the students should do the following steps:

i. Using any text editor, open Authors.xml and create the document type declaration.

ii. Create element declarations for all of the elements.

   a. The top-most element in the document is the AUTHORS element may contain one or more occurrences.

   b. The AUTHOR element has 3 child elements: AUTHOR_Name, PHONE, and books. Remember that PHONE is optional.

   c. AUTHOR_Name has 2 child elements: First, and Last.
   d. The First, Last, and Phone elements can only contain character data.
   e. Next, add declaration for the books element, which must contain at least one occurrence of the book element.
   f. Each book has 2 child elements: book name, and Publisher. Both of them contain character information.

iii. Insert the attribute declarations right below the declaration for the element that contains the attribute.
   a. Insert the declaration for AUTHORID (required element).
   b. Title is an optional element that can have one of three possible values: Mr., Mrs., or Ms.
   c. Add the attribute declaration for bookID (required).
   d. Insert the attribute declaration for version. This is an optional element. The default value is "1.1".

iv. When you created a DTD, save your file with the same name.

v. Open your XML file in XML Spy and validate your document against the rules you set up in the DTD.

vi. If you get an error message, examine the reason for the error and then make the necessary corrections.

vii. After confirming that Authors.xml is a valid document, close XML Spy.

7.0 REFERENCES/FURTHER READING

Charles, F. G., Paul, P. (1998). "The XML Handbook", Prentice Hall Computer Books.

Elliotte, R. H. & Scott, M. W. (2001). "XML in a Nutshell".

.Michael, M. (2001). "HTML and XML for Beginners".

UNIT 3 XML SCHEMA BASICS

CONTENTS

1.0 INTRODUCTION

This unit introduces an XML Schema which is an XML document that defines the content and structure of one or more XML documents. An XML schema defines element and attribute names for a class of XML documents. The schema also specifies the structure that those documents must adhere to and the type of content that each element can hold.

2.0 OBJECTIVES

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

- identify the purpose of XML Schema.
- state the limitations of DTDs
- explain the power of XML Schema.
- describe how to validate an XML Instance with an XML schema.

3.0 MAIN CONTENT

The Purpose of XML Schema
XML Schema is an XML-based language used to create XML-based languages and data models. An XML schema defines element and attribute names for a class of XML documents. The schema also specifies the structure that those documents must adhere to and the type of content that each element can hold.

XML documents that attempt to adhere to an XML schema are said to be instances of that schema. If they correctly adhere to the schema, then they are valid instances. This is not the same as being well formed. A well-formed XML document follows all the syntax rules of XML, but it does necessarily adhere to any particular schema. So, an XML document can be well formed without being valid, but it cannot be valid unless it is well formed.

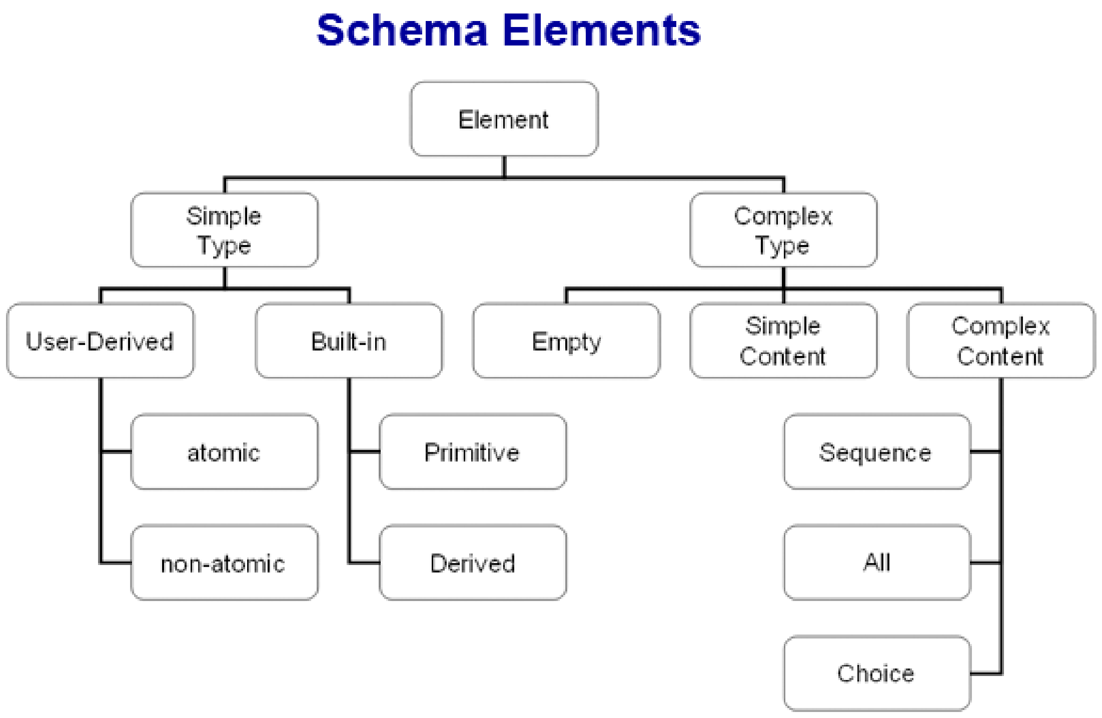
3.1 The Power of XML Schema

You may already have some experience with DTDs. DTDs are similar to XML schemas in that they are used to create classes of XML documents. DTDs were around long before the advent of XML. They were originally created to define languages based on SGML, the parent of XML. Although DTDs are still common, XML Schema is a much more powerful language.

As a means of understanding the power of XML Schema, let's look at the limitations of DTD:

a. DTDs do not have built-in datatypes
b. DTDs do not support user-derived datatypes
c. DTDs allow only limited control over cardinality (the number of occurrences of an element within its parent)
d. DTDs do not support Namespaces or any simple way of reusing or importing other schemas.

An XML schema describes the structure of an XML instance document by defining what each element must or may contain. An element is limited by its type. For example, an element of complex type can contain child elements and attributes, whereas a simple-type element can only contain text. The diagram below gives a first look at the types of XML Schema elements.



Schema authors can define their own types or use the built-in types. Throughout this course, we will refer back to this diagram as we learn to define elements. You may want to bookmark this page, so that you can easily reference it.

The following is a high-level overview of Schema types:

- elements can be of simple type or complex type
- simple type elements can only contain text. they can not have child elements or attributes
- all the built-in types are simple types (e.g, xs:string)
- schema authors can derive simple types by restricting another simple type. for example, an email type could be derived by limiting a string to a specific pattern
- simple types can be atomic (e.g. strings and integers) or non-atomic (e.g. lists)
- complex-type elements can contain child elements and attributes as well as text
- by default, complex-type elements have complex content, meaning that they have child elements
- complex-type elements can be limited to having simple content, meaning they only contain text. they are different from simple type elements in that they have attributes
- complex types can be limited to having no content, meaning they are empty, but they may have attributes

- complex types may have mixed content - a combination of text and child elements.

SELF-ASSESSMENT EXERCISE

i. What is an XML Schema?
ii. State the limitation of DTD over XML Schema.

3.2 A Simple XML Schema

Let's take a look at a simple XML schema, which is made up of one complex type element with two child simple type elements.

- Code Sample: SchemaBasics /Author.xsd
- <?xml version="1.0" encoding="UTF-8"?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
- <xs:element name="Author">
- <xs:complexType>
- <xs:sequence>
- <xs:element name="FirstName" type="xs:string" />
- <xs:element name="LastName" type="xs:string" />
- </xs:sequence>
- </xs:complexType>
- </xs:element>
- </xs:schema>

Code Explanation
As you can see, an XML schema is an XML document and must follow all the syntax rules of any other XML document; that is, it must be well formed. XML schemas also have to follow the rules defined in the "Schema of schemas," which defines, among other things, the structure of an element and attribute names in an XML schema.

Although it is not required, it is a common practice to use the xs qualifier to identify schema elements and types.

The document element of XML schemas is xs:schema. It takes the attribute xmlns:xs with the value of http://www.w3.org/2001/XMLSchema, indicating that the document should follow the rules of XML Schema. This will be clearer after you learn about namespaces.

In this XML schema, we see a xs:element element within the xs:schema element. xs:element is used to define an element. In this case it defines the element author as a complex type element, which contains a sequence of two elements: FirstName and LastName, both of which are of the simple type, string.

3.3 Validating an XML Instance Document

In the section 3.2, you saw an example of a simple XML schema, which defined the structure of an Author element. The code sample below shows a valid XML instance of this XML schema.

Code Sample: SchemaBasics/MarkTwain.xml

- <?xml version="1.0"?>
- <Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
- xsi:noNamespaceSchemaLocation="Author.xsd">
- <FirstName>Mark</FirstName>
- <LastName>Twain</LastName>
- </Author>

Code Explanation
This is a simple XML document. Its document element is Author, which contains two child elements: FirstName and LastName, just as the associated XML schema requires.

The xmlns:xsi attribute of the document element indicates that this XML document is an instance of an XML schema. The document is tied to a specific XML schema with the xsi:noNamespaceSchemaLocation attribute.

There are many ways to validate the XML instance. If you are using an XML authoring tool, it very likely is able to perform the validation for you. Alternatively, a couple of simple online XML Schema validator tools are listed below.

- http://tools.decisionsoft.com/schemaValidate.html provided by DecisionSoft.
- http://apps.gotdotnet.com/xmltools/xsdvalidator provided by GotDotNet.

SELF-ASSESSMENT EXERCISE

i. Differentiate between a simple type XML schema and a complex type XML schema
ii. Modify a simple XML schema in section 3.2 which made up of one complex type with three child simple type elements.

iii. Validate your XML schema you created by following the procedure in section 3.3.

4.0 CONCLUSION

XML Schema is an XML document that defines the content and structure of one or more XML documents and defines element and attribute names for a class of XML documents. The schema also specifies the structure that those documents must adhere to and the type of content that each element can hold.

5.0 SUMMARY

In this unit, you have learned to create a very simple XML Schema and to use it to validate an XML instance document. You are now ready to learn more advanced features of XML Schema in the next unit.

6.0 TUTOR-MARKED ASSIGNMENT

James created two documents named News.xml and NewsDisplay.htm. He has already created the basic format for the Web page. He needs your help in binding a single record from News.xml file with the Web page and binding the XML elements to the HTML tags. The Web page should display the news title, author's name and one paragraph with news text.

NewsDisplay.htm

```
<html>
<head>
<!--NewsDisplay.htm -->
<title>News Display</title>
<style>
body {background-colour:rgb(255,255,192)}
#News_Title (font-family:Ariel, Helvetica, sans-serif; font-size:40pt)
#News_Author (font-family:Ariel, Helvetica, sans-serif)
#News_Text (font-family:Ariel, Helvetica, sans-serif)
</style>
</head>

<body>
<div id="News_Text"> Add your name and the date here.</div>
<div id="News_Title">Title</div>
<div id="News_Author">Author Name</div>
<div id="News_Text">Text</div>
</body>
```

```
</html>
```

News.xml

```
<?xml version ="1.0"?>
<!--News.xml-->

<Information>
        <Title> Recognise the components of the Visual Basic Editor
</Title>
        <Author>Garry Gibson </Author>
<Newsdata>

<![CDATA[
Visual Basic for Applications is an excellent built-in tool for
customising or automating operations within the Microsoft Office
application packages as well as many non-Microsoft software packages.
The Visual Basic Editor is actually a set of three windows: the main, the
Project Explorer and Properties windows. The main window comprised
of the title bar, menu bar and the standard toolbar appears at the top of
the screen. The properties window includes characteristics for each
object controlling its appearance and behaviour. Within the Project
explorer window is the code window that is opened whenever VBA
instructions called a procedure are entered as a block of code.
]]>
</Newsdata>

</Information>
```

To complete the project students should take the following steps:

i. Using any text editor, open News.xml and NewsDisplay.htm. Take some time to view these documents, becoming familiar with their contents and structures.
ii. Insert your name and the date into NewsDisplay.htm for display on a Web page.
iii. In NewsDisplay.htm, create data island “News_Info” that points to the file News.xml.
iv. Bind the page title, author, and text to the Title, Author, and Newsdata of the News_Info data island.
v. Save the changes to the file.
vi. Using Internet Explorer, open NewsDisplay.htm and verify that it shows all the information from News.xml data source.
vii. Correct any syntax/logical errors, save your file and verify the content with Internet Explorer.
viii. Print NewsDisplay.htm and the web page, generated by Internet Explorer.

7.0 REFERENCES/FURTHER READING

Charles, F. G.,& Paul, P. (1998). “The XML Handbook”, Prentice Hall Computer Books.

Elliotte, R. H. & Scott, M. W. (2001). “XML in a Nutshell”.

Michael, M. (2001). “HTML and XML for Beginners”.

UNIT 4 XSLT BASICS

CONTENTS

1.0 INTRODUCTION

In 1998, the W3C developed the Extensible Style sheet Language, or XSL. XSL is composed of XSL-FO (Extensible Style sheet Language – Formatting Objects) and XSLT (Extensible Style sheet Language Transformations).

XSL-FO is used to implement page layout and design XSLT is used to transform XML content into another presentation format. An XSLT style sheet contains instructions for transforming the contents of an XML document into another format.

2.0 OBJECTIVES

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

- explain the purpose of XSLT.
- design a simple XSLT transform.
- explain and work with whitespace
- describe how to choose the output mode
- describe how to output elements and attributes.

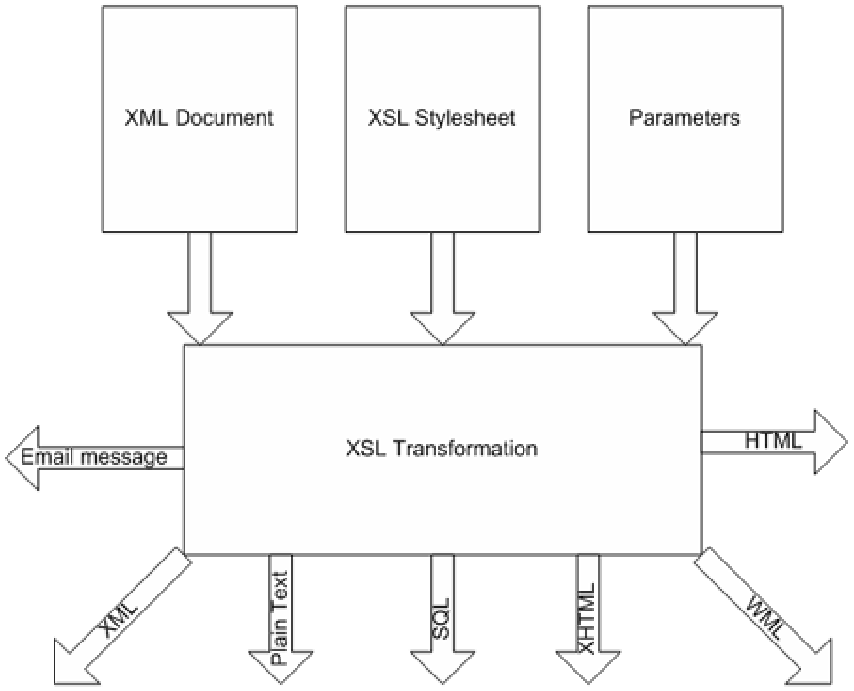
3.0 MAIN CONTENT

eXtensible Stylesheet Language

The eXtensible Stylesheet Language is divided into two sub-languages: eXtensible Stylesheet Language Transformations (XSLT) and eXtensible Stylesheet Language - Formatting Objects (XSL-FO). In this lesson, we will be looking at the basics of XSLT, which is used to transform XML documents.

XSLT documents are well-formed XML documents that describe how another XML document should be transformed. For XSLT to work, it needs an XML document to transform and an engine to make the transformation take place. In addition, parameters can be passed in to XSLTs providing further instructions on how to do the transformation.

The diagram below shows how this all works.



3.1 The Transformation Process

An XSLT looks at an XML document as a collection of nodes of the following types:

1. Root node
2. Element nodes
3. Attribute nodes
4. Text nodes

5. Processing instruction nodes
6. Comment nodes.

An XSLT document contains one or more templates, which are created with the <xsl:template /> tag. When an XML document is transformed with an XSLT, the XSLT processor reads through the XML document starting at the root, which is one level above the document element, and progressing from top to bottom, just as a person would read the document. Each time the processor encounters a node, it looks for a matching template in the XSLT.

If it finds a matching template it applies it; otherwise, it uses a default template as defined by the XSLT specification. The default templates are shown in the table below.

Default Templates	
Node Type	**Default Template**
Root	Apply templates for child nodes.
Element	Apply templates for child nodes.
Attribute	Output attribute value.
Text	Output text value.
Processing Instruction	Do nothing.
Comment	Do nothing.

In this context, attributes are not considered children of the elements that contain them, so attributes get ignored by the XSLT processor unless they are explicitly referenced by the XSLT document.

3.2 An XSLT StyleSheet

Let's start by looking at a simple XML document and an XSLT styleSheet, which is used to transform the XML to HTML.

Code Sample: ayodele.xml

- <?xml version="1.0"?>
- **<?xml-stylesheet href="beatle.xsl" type="text/xsl"?>**
- <person>
- <name>
- <firstname>Ayodele</firstname>
- <lastname>Oyelade</lastname>
- </name>
- <job>Singer</job>
- <gender>Female</gender>

- </person>

Code Explanation
This is a straightforward XML document. The processing instruction at the top indicates that the XML document should be transformed using beatle.xsl (shown below).

Code Sample:beatle.xsl

- <?xml version="1.0"?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"><xsl:output method="html"/>
- <xsl:template match="child::person">
- <html>
- <head>
- <title>
- <xsl:value-of select="descendant::firstname" />
- <xsl:text> </xsl:text>
- <xsl:value-of select="descendant::lastname" />
- </title>
- </head>
- <body>
- <xsl:value-of select="descendant::firstname" />
- <xsl:text> </xsl:text>
- <xsl:value-of select="descendant::lastname" />
- </body>
- </html>
- </xsl:template>
- </xsl:stylesheet>

Code Explanation
Note that the document begins with an XML declaration. This is because XSLTs are XML documents themselves. As with all XML documents, the XML declaration is optional.

The second line (shown below) is the document element of the XSLT. It states that this document is a version 1.0 XSLT document.

```
<xsl:stylesheetversion="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

The third line (shown below) indicates that the resulting output will be HTML.

```
<xsl:output method="html"/>
```

The fourth line is an open <xsl:template> element. The match attribute of this tag takes an XPath, which indicates that this template applies to the person node of the XML document. Because person is the document element of the source document, this template will only run once.

There are then a few lines of HTML tags followed by two <xsl:value-of /> elements separated by one <xsl:text> element. The <xsl:value-of /> tag has a select attribute, which takes an XPath pointing to a specific element or group of elements within the XML document. In this case, the two <xsl:value-of /> tags point to firstname and lastname elements, indicating that they should be output in the title of the HTML page. The <xsl:text> element is used to create a space between the first name and the last name elements.

```
<xsl:value-of select="descendant::firstname" />
<xsl:text> </xsl:text>
<xsl:value-of select="descendant::lastname" />
```

There are then some more HTML tags followed by the same XSLT tags, re-outputting the first and last name of the Beatle in the body of the HTML page.

3.3 xsl:template

The xsl:template tag is used to tell the XSLT processor what to do when it comes across a matching node. Matches are determined by the XPath expression in the match attribute of the xsl:template tag.

Code Sample: XsltBasics/FirstName.xsl

- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="text"/>
- **<xsl:template match="firstname">**
- **We found a first name!**
- **</xsl:template>**
- </xsl:stylesheet>

Code Explanation
If we use this XSLT to transform firstname.xml, which has the same XML as ayodele.xml , the result will be:

- *We found a first name!*
- *OyeladeSingerMale*

The text "We found a first name!" shows up only once, because only one match is found. The text "OyeladeSingerMale" shows up because the XSLT engine will continue looking through an XML document until it is explicitly told to stop. When it reaches a text node with no matching template, it uses the default template, which simply outputs the text. "Oyelade", "Singer" and "Male" are the text node values inside the elements that follow the FirstName element.

Let's see what happens if we transform an XML document with multiple FirstName elements against this same XSLT. Take, for example, the XML document below.

Code Sample:beatles.xml

```
•	<?xml version="1.0"?>
•	<?xml-stylesheet href="firstName.xsl" type="text/xsl"?>
•	<beatles>
•	<beatle link="http://www.ayodeleoyelade.com">
•	<name>
•	<firstname>Ayodele</firstname>
•	<lastname>Oyelade</lastname>
•	</name>
•	</beatle>
•	<beatle link="http://www.olanreoyelade.com">
•	<name>
•	<firstname>Olanrewaju</firstname>
•	<lastname>Oyelade</lastname>
•	</name>
•	</beatle>
•	<beatle link="http://www.peculiaroyelade.com">
•	<name>
•	<firstname>Peculiar</firstname>
•	<lastname>Oyelade</lastname>
•	</name>
•	</beatle>
•	<beatle link="http://www.adamsmith.com">
•	<name>
•	<firstname>Adam</firstname>
•	<lastname>Smith</lastname>
•	</name>
•	</beatle>
•	<beatle link="http://www.openuniversity.com" real="no">
•	<name>
•	<firstname>Nat</firstname>
•	<lastname>Dunn</lastname>
```

- </name>
- </beatle>
- </beatles>

Code Explanation
The resulting output will be as follows.

a. We found a first name! Oyelade
b. We found a first name! Oyelade
c. We found a first name! Oyelade
d. We found a first name! Smith
e. We found a first name! Dunn

Each time a firstname element is found in the source XML document, the text "We found a first name!" is output. For the other elements with text (in this case, only lastname elements), the actual text is output.

SELF-ASSESSMENT EXERCISE

i. Mention the types of eXtensible Stylesheet Language
ii. Write short notes on the following:
 a. XSLT StyleSheet
 b. xsl:template
 c. xsl:value-of
 d. XPath.

3.4 xsl:value-of

The xsl:value-of element is used to output the text value of a node. To illustrate, let's look at the following example.

Code Sample: valueo11.xsl
- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="text"/>
- <xsl:template match="firstname">
- We found a first name and it's <xsl:value-of select="."/>
- </xsl:template>
- </xsl:stylesheet>

Code Explanation
If we use this XSLT to transform beatles-vo1.xml, which has the same XML as beatles.xml, the result will be:

a. We found a first name and it's AyodeleOyelade
b. We found a first name and it's OlanrewajuOyelade
c. We found a first name and it's PeculiarOyelade
d. We found a first name and it's AdamSmith
e. We found a first name and it's NatDunn

The select attribute takes an XPath, which is used to indicate which node's value to output. A single dot (.) refers to the current node (i.e. the node that was matched by the template).

Don't let the last names at the end of each line confuse you. These are not part of the output of the xsl:value-of element. They are there as a result of the default template, which outputs the text value of the element found.

To illustrate this, let's look at another example.

Code Sample: XsltBasics/ValueOf2.xsl

- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="text"/>
- <xsl:template match="firstname">
- We found a first name and it's <xsl:value-of select="."/>
- </xsl:template>
- <xsl:template match="lastname">
- We found a last name and it's <xsl:value-of select="."/>
- </xsl:template>
- </xsl:stylesheet>

Code Explanation
If we use this XSLT to transform beatles-VO2.xml, which has the same XML as beatles.xml , the result will be:

a. We found a first name and it's Ayodele
b. We found a last name and it's Oyelade
c. We found a first name and it's Olanrewaju
d. We found a last name and it's Oyelade
e. We found a first name and it's Peculiar
f. We found a last name and it's Oyelade
g. We found a first name and it's Adam
h. We found a last name and it's Smith
i. We found a first name and it's Nat
j. We found a last name and it's Dunn

This XSLT has a template for both firstname and lastname and so it never uses the default template.

3.5 Whitespace and xsl:text

Whitespace in an XSLT template is output literally. If you're not careful, this can lead to undesirable results. To illustrate, let's look at the following XML and XSLT documents.

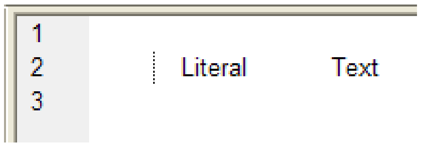
Code Sample: whitespace1.xml

```
•	<?xml version="1.0" encoding="UTF-8"?>
•	<?xml-stylesheet type="text/xsl" href="WhiteSpace1.xsl"?>
•	<example>
•	<blurb>
•	Hello World!
•	</blurb>
•	</example>
```

Code Sample: whitespace1.xsl

```
•	<?xml version="1.0" encoding="UTF-8"?>
•	<xsl:stylesheet version="1.0"
•	xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
•	<xsl:output method="text"/>
•	<xsl:template match="blurb">
•	Literal   Text
•	</xsl:template>
•	</xsl:stylesheet>
```

When whitespace1.xml is transformed against whitespace1.xsl, the output looks like this:



```

		Literal		Text

```

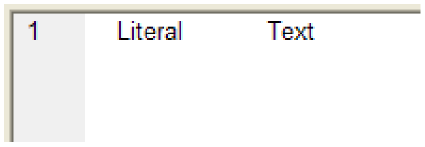
There is an empty line before and after the text. There are also two tabs preceding the text. This is because the whitespace between <xsl:template match="blurb"> and Literal Text and the whitespace between Literal Text and </xsl:template> is output literally.

If you did not want that extra whitespace to show up in the output, you could use the following XSLT instead.

Code Sample: WhiteSpace2.xsl

- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="text"/>
- <xsl:template match="blurb">
- **<xsl:text>**Literal   Text**</xsl:text>**
- </xsl:template>
- </xsl:stylesheet>

Code Explanation
When WhiteSpace2.xml, which has the same XML as WhiteSpace1.xml , is transformed against WhiteSpace2.xsl, the output looks like this:



```
1    Literal        Text
```

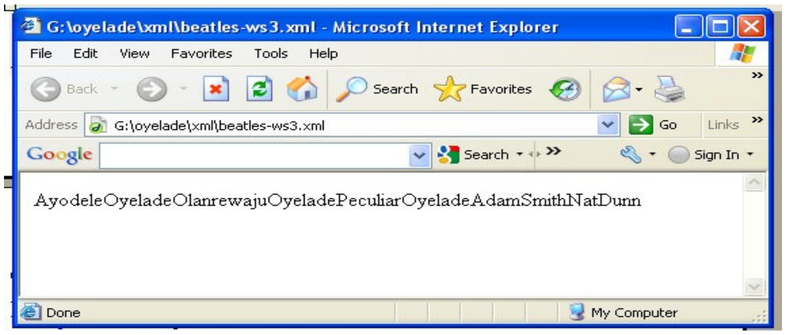
Because whitespace between open tags (e.g, between <xsl:template match="blurb"> and <xsl:text>) and whitespace between close tags (e.g, </xsl:text> and </xsl:template>) is ignored, the only content that is output is the text between the open and close xsl:text tags.

3.6 Inserting Whitespace with xsl:text

The examples above illustrate how xsl:text can be used to remove whitespace. It can also be used to add whitespace where there otherwise wouldn't be any. Let's take a look at another example.

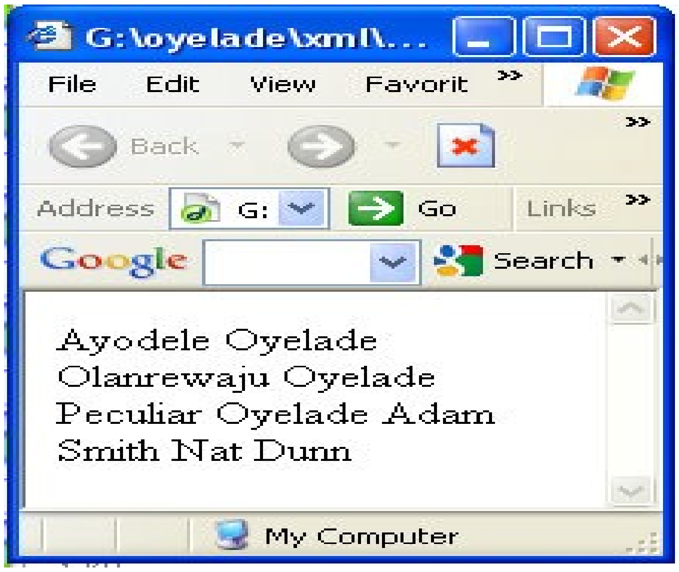
Code Sample:WhiteSpace3.xsl
- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="text"/>
- <xsl:template match="name">
- **<xsl:value-of select="firstname"/>**
- **<xsl:value-of select="lastname"/>**
- </xsl:template>
- </xsl:stylesheet>

Code Explanation
When Beatles-WS3.xml, which has the same XML as Beatles.xml , is transformed against WhiteSpace3.xsl, the output looks like this:


G:\oyelade\xml\beatles-ws3.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address G:\oyelade\xml\beatles-ws3.xml Go Links
Google Search Sign In
Done
My Computer

```
AyodeleOyeladeOlanrewajuOyeladePeculiarOyeladeAdamSmithNatDunn
```

Clearly, this isn't the desired output. We'd like to have each Beatle on a separate line and spaces between their first and last names like this:



```
Ayodele Oyelade
Olanrewaju Oyelade
Peculiar Oyelade Adam
Smith Nat Dunn
```

However, because whitespace between two open tags and between two close tags is ignored, we don't get any whitespace in the output. We can use xsl:text to fix this by explicitly indicating how much whitespace we want in each location.

Code Sample: WhiteSpace4.xsl

- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="text"/>
- <xsl:template match="name">
- <xsl:value-of select="firstname"/>
- **<xsl:text> </xsl:text>**
- <xsl:value-of select="lastname"/>
- **<xsl:text>**
- **</xsl:text>**

- </xsl:template>
- </xsl:stylesheet>

Code Explanation
Notice that the second close xsl:text hugs the left border. If we were to put any space before it on that line, that space would show up in the output. To see the desired result, transform Beatles-WS4.xml against WhiteSpace4.xsl.

3.7 Output Types

The type of output is determined by the method attribute of the xsl:output element, which must be a direct child of the xsl:stylesheet element.

- Text
- <xsl:output method="text"/>

As we have seen in the examples thus far, XSLT can be used to output plain text. However, XSLT is much more commonly used to transform one XML structure into another XML structure or into HTML.

- XML
- <xsl:output method="xml"/>

The default output method (in most cases) is XML. That is, an XSLT with no xsl:output element will output XML. In this case, XSLT tags are intermingled with the output XML tags. Generally, the XSLT tags are prefixed with xsl:, so it is easy to tell them apart. The output XML tags may also be prefixed, but not with the same prefix as the XSLT tags.

3.7.1 Literal Result Elements

The output XML tags are called literal result elements. That is because they are output literally rather than used by the XSLT to determine what or how something should be output.

Useful xsl:output Attributes When Outputting XML	
Attribute	**Description**
Method	Should be set to xml.
Indent	Yes or no. If yes, the resulting XML document will be pretty printed. The default is usually no.
Omit-xml-declaration	Yes or no. If no, the resulting XML document will begin with an XML declaration. The default is usually no.
Version	Specifies the XML version of the resulting XML document.

The documents below show how XSLT can be used to output XML.

Code Sample: Name.xml

```
•   <?xml version="1.0"?>
•   <?xml-stylesheet href="Name.xsl" type="text/xsl"?>
•   <person>
•   <name>
•   <firstname>Ayodele</firstname>
•   <lastname>Oyelade</lastname>
•   </name>
•   </person>
•   Code Sample: Name.xsl
•   <?xml version="1.0" encoding="UTF-8"?>
•   <xsl:stylesheet version="1.0"
•   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
•   <xsl:output method="xml" indent="yes" version="1.0"/>
•   <xsl:template match="name">
•   <Matches>
•   <Match>We found a name!</Match>
•   <Name><xsl:value-of select="."/></Name>
•   </Matches>
•   </xsl:template>
•   </xsl:stylesheet>
•   Code Sample: NameResult.xml
•   <?xml version="1.0" encoding="UTF-8"?>
•   <Matches>
•   <Match>We found a name!</Match>
•   <Name>AyodeleOyelade</Name>
•   </Matches>
```

3.7.2 HTML

<xsl:output method="html"/>
It is very common to use XSLT to output HTML. In fact, XSLT even has a special provision for HTML: if the document element of the resulting document is html (not case sensitive) then the default method is changed to HTML. However, for the sake of clarity, it is better code to specify the output method with the xsl:output tag.

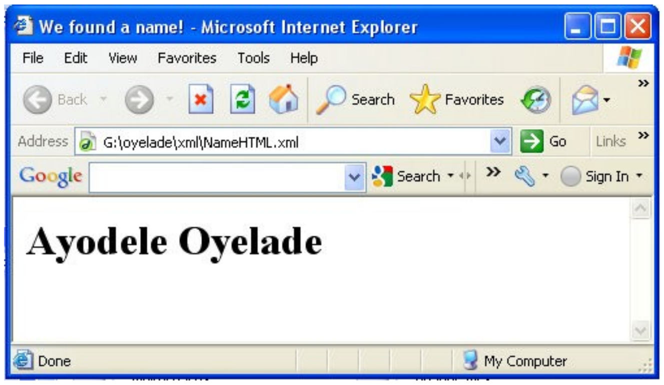
The documents below show how XSLT can be used to output HTML.

Code Sample: NameHTML.xml

- <?xml version="1.0"?>
- <?xml-stylesheet href="NameHTML.xsl" type="text/xsl"?>
- <person>
- <name>
- <firstname>Ayodele</firstname>
- <lastname>Oyelade</lastname>
- </name>
- </person>
- Code Sample: NameHTML.xsl
- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- **<xsl:output method="html"/>**
- <xsl:template match="name">
- <html>
- <head>
- <title>We found a name!</title>
- </head>
- <body>
- <h1>
- <xsl:value-of select="firstname"/>
- <xsl:text> </xsl:text>
- <xsl:value-of select="lastname"/>
- </h1>
- </body>
- </html>
- </xsl:template>
- </xsl:stylesheet>

Code Explanation

If you open NameHTML.xml in Internet Explorer, you'll see the following result.


We found a name! - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address G:\oyelade\xml\NameHTML.xml Go Links
Google Search Sign In
Done My Computer

Ayodele Oyelade

3.8 Elements and Attributes

When outputting XML or HTML, you will need to output tags and attributes. We saw earlier that you can output tags using literal result elements.

xsl:element

The xsl:element tag can be used to explicitly specify that an element is for output. You will notice that the xsl:element tag takes the name attribute, which is used to specify the name of the element being output. Also, as shown, xsl:element tags can be nested within each other.

Code Sample: Name2.xsl

- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="xml" indent="yes" version="1.0"/>
- <xsl:template match="name">
- **<xsl:element name="Matches">**
- **<xsl:element name="Match">We found a name!</xsl:element>**
- **<xsl:element name="Name">**
- **<xsl:value-of select="."/>**
- **</xsl:element>**
- **</xsl:element>**
- </xsl:template>
- </xsl:stylesheet>

When Name2.xml, which has the same XML as Name.xml , is transformed against Name2.xsl, the output looks the same as we saw when we used literal result elements earlier in the lesson.

The xsl:element tag is most useful when the tag name itself is generated. If the tag name is not generated, it is generally easier to use literal result elements.

xsl:attribute
The xsl:attribute element is similar to the xsl:element element. It is used to explicitly output an attribute. However, it is more commonly used than xsl:element. To see why, let's first look at an example in which an attribute is output literally.

Code Sample: NameLREwithAtt.xsl
- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="xml" indent="yes" version="1.0"/>
- **<xsl:template match="name">**
- **<Matches>**
- **<Match Name="Some Name">We found a name!</Match>**
- **</Matches>**
- **</xsl:template>**
- </xsl:stylesheet>

Code Explanation
Notice that the value of the Name attribute is just "Some Name". We would like to generate this value from the source XML document by doing something like this:

- <!--THIS IS POORLY FORMED-->
- <Match Name="<xsl:value-of select='.'/>">
- We found a name!
- </Match>
- Code Sample: NameWithAtt.xsl
- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="xml" indent="yes" version="1.0"/>
- <xsl:template match="name">
- <Matches>
- <Match>
- **<xsl:attribute name="Name">**
- **<xsl:value-of select="firstname"/>**

- **<xsl:text> </xsl:text>**
- **<xsl:value-of select="lastname"/>**
- **</xsl:attribute>**We found a name!</Match>
- </Matches>
- </xsl:template>
- </xsl:stylesheet>

Code Explanation

The xsl:attribute tag applies to the element that it is nested within; in this case, the match element. When NameWithAtt.xml, which has the same XML as Name.xml, is transformed against NameWithAtt.xsl, the output looks like this:

Code Sample: NameWithAttResult.xml

- <?xml version="1.0" encoding="UTF-8"?>
- <Matches>
- <Match Name="AyodeleOyelade">We found a name!</Match>
- </Matches>

3.8.1 Attributes and Curly Brackets

Because using xsl:attribute can be a bit laborious, an abbreviated syntax is available. An XPath within curly brackets ({}) can be embedded as the value of an attribute in the open tag. The following example illustrates this.

Code Sample: NameWithAttAbbr.xsl

- <?xml version="1.0" encoding="UTF-8"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:output method="xml" indent="yes" version="1.0"/>
- <xsl:template match="name">
- <Matches>
- <Match Name="**{firstname} {lastname}**">We found a name!</Match>
- </Matches>
- </xsl:template>
- </xsl:stylesheet>
- Code Explanation

This will have the exact same result as NameWithAtt.xsl , but it is much quicker and easier to write. To see the result, transform NameWithAttAbbr.xml against this XSLT.

SELF-ASSESSMENT EXERCISE

i. **XML**

Start by creating a well-formed XML file, such as books.xml (as shown below) in your text editor:

- <!-- This is books.xml -->
- <?xml version="1.0" encoding="ISO-8859-1"?>
- <bib>
- <book isbn="1112223330">
- <year>1994</year>
- <title>TCP/IP Illustrated</title>
- <author>
- <last>Stevens</last>
- <first>W.</first>
- </author>
- <publisher>Addison-Wesley</publisher>
- <price>65.95</price>
- </book>
- <book isbn="1112223331">
- <year>1992</year>
- <title>Advanced Programming in the Unix environment</title>
- <author><last>Stevens</last><first>W.</first></author>
- <publisher>Addison-Wesley</publisher>
- <price>65.95</price>
- </book>
- <book isbn="1112223332">
- <year>2000</year>
- <title>Data on the Web</title>
- <author>
- <last>Abiteboul</last>
- <first>Serge</first>
- </author>
- <author>
- <last>Buneman</last>
- <first>Peter</first>
- </author>
- <author>
- <last>Suciu</last>
- <first>Dan</first>
- </author>
- <publisher>Morgan Kaufmann Publishers</publisher>
- <price>39.95</price>
- </book>

- <book isbn="1112223333">
- <year>1999</year>
- <title>The Technology and Content for Digital TV</title>
- <editor>
- <last>Gerbarg</last>
- <first>Darcy</first>
- <affiliation>CITI</affiliation>
- </editor>
- <publisher>Kluwer Academic Publishers</publisher>
- <price>129.95</price>
- </book>
- </bib>

ii. **XSL**

Create an XSL file (books.xsl) as shown below:

- </xsl:stylesheet>
- <?xml version="1.0" encoding="ISO-8859-1"?>
- <xsl:stylesheet version="1.0"
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:template match="/">
- <html>
- <html>
- <head>
- <style type="text/css">
- table {font-family: arial}
- </style>
- </head>
- <body>
- <h2>My Favorite Books</h2>
- <table>
- <tr bgcolour="#9acd32">
- <th>Title</th>
- <th>Author</th>
- <th>Price</th>
- </tr>
- <xsl:for-each select="bib/book">
- <tr>
- <td><xsl:value-of select="title"/></td>
- <xsl:choose>
- <xsl:when test="price &lt; 50">
- <td bgcolour="#ff00ff">
- <xsl:value-of select="author"/>
- </td>
- </xsl:when>
- <xsl:when test="price &gt; 50 and price &lt; 100">

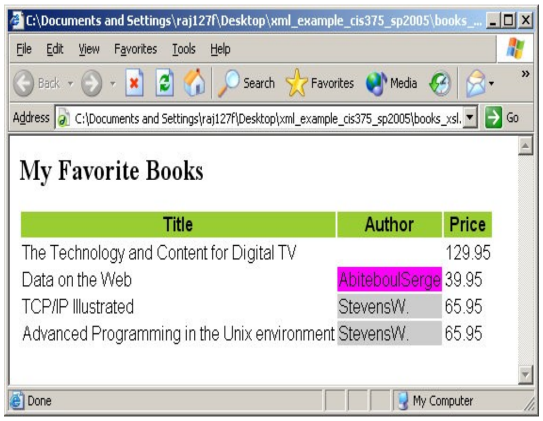
- <td bgcolour="#cccccc">
- <xsl:value-of select="author"/></td>
- </xsl:when>
- <xsl:otherwise>
- <td><xsl:value-of select="author"/></td>
- </xsl:otherwise>
- </xsl:choose>
- <td><xsl:value-of select="price"/></td>
- </tr>
- </xsl:for-each>
- </table>
- </body>
- </html>
- </xsl:template>

Add the following line of code to the books.xml file in 1 above (after the line
<?xml version="1.0" encoding="ISO-8859-1"?>) and resave the file as books_xsl.xml:

<!DOCTYPE note SYSTEM "books.xsl ">

iii. Validate the program books.xsl which is used to display the contents of an XML file (books_xsl.xml)

The output should look like the one shown below:


C:\Documents and Settings\raj127f\Desktop\xml_example_cis375_sp2005\books_...
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address C:\Documents and Settings\raj127f\Desktop\xml_example_cis375_sp2005\books_xsl.
Go
Done
My Computer

My Favorite Books

Title	Author	Price
The Technology and Content for Digital TV		129.95
Data on the Web	AbiteboulSerge	39.95
TCP/IP Illustrated	StevensW.	65.95
Advanced Programming in the Unix environment	StevensW.	65.95

4.0 CONCLUSION

XSL is composed of XSL-FO (Extensible Style sheet Language – Formatting Objects) and XSLT (Extensible Style sheet Language Transformations). XSL-FO is used to implement page layout and design and XSLT is used to transform XML content into another presentation format. An XSLT style sheet contains instructions for transforming the contents of an XML document into another format.

5.0 SUMMARY

In this unit, you have learned the basics of creating XSLTs to transform XML instances into text, HTML and other XML structures.

6.0 TUTOR-MARKED ASSIGNMENT

Project.xml into HTML document. The result document should display Student’s name using the <h1> tag, date using the <h2> tag, the objective name using the <h4> tag, and the description using the <div> tag. Display it on a web browser and change the appearance of the file by modifying the XSLT style sheet.

Project.xml

```
<?xml version="1.0"?>
<!-- XML Project --> Janet has created an XML document
Project.xml. Janet asks students to create an XSLT style sheet style.css
to transform

<project>
<student>Student</student>
<date>Date</date>
<objective>
<name>Working with XSL</name>
<description>XSL is composed of three parts: XSL-FO
(Extensible Style sheet Language - Formatting Objects),
XSLT (Extensible Style sheet Language Transformations),
and XPath.  XSL-FO is used to implement page layout and design.
XSLT is used to transform XML content into another presentation
format.
XPath is used to locate information from an XML document and
perform operations and calculations upon that content.
</description>
</objective>
<objective>
<name>Introducing XSLT style sheets and processors</name>
<description>An XSLT style sheet contains instructions for
transforming the contents of an XML document into another format. An
XSLT style sheet document is itself an XML document, but has an
extension .xsl.
An XSLT style sheet converts a source document of XML content into
 a result document containing the markup codes and other instructions
for formatting.
</description>
</objective>
```

```
<objective>
<name>Creating an XSLT style sheet</name>
<description>To attach an XML file to the style sheet, insert the processing instruction following the first line in the document. An XSLT style sheet has the general structure of all XML documents.
</description>
</objective>
</project>
```

To complete this task, the students should do the following steps:

i. To attach an XSLT style sheet to the XML document, add the processing instruction to the XML file, save it as **Projectstyle.xml**, and close.
ii. Using any text editor, create a new XSLT style sheet document **style.css**.
iii. Create the general structure of an XSLT style sheet.
iv. Create the root template.
v. Specify the output method to transform the source document into an HTML document.
vi. Insert node values into the result document.
vii. Open **Projectstyle.xml** using your browser and verify that it displays the result document correctly.
viii. Print all your documents and the result document view generated by the browser.

7.0 REFERENCES/FURTHER READING

Charles, F. G.,& Paul, P. (1998). “The XML Handbook”, Prentice Hall Computer Books.

Elliotte, R. H. & Scott, M. W. (2001). “XML in a Nutshell”.

Michael, M. (2001). “HTML and XML for Beginners”.

UNIT 5 MICROMEDIA DREAMWEAVER

CONTENTS

1.0 INTRODUCTION

Dreamweaver is a web page editor, for creating web pages. It includes many new features that help you build websites and applications with a minimal amount of time and effort. Dreamweaver makes complex technologies simple and accessible, helping you accomplish more in less time. Additional features were added to Dreamweaver 8 from its predecessor Dreamweaver MX 2004.

Following are a few of the key new features in Dreamweaver 8 such as Zoom tool and guides, Visual XML data binding, New CSS Styles panel, CSS layout visualisation, Code collapse, Coding toolbar, Background file transfer, and Insert Flash Video command.

2.0 OBJECTIVES

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

- describe how to get started with Dreamweaver 8.
- describe the opening and saving documents in Dreamweaver 8.
- state Dreamweaver workspace elements
- describe how to work with Dreamweaver
- create new files in Dreamweaver
- explain opening and saving files in Dreamweaver

3.0 MAIN CONTENT

How to Get Started With Dreamweaver 8

From the Start button normally located in the bottom left corner of the screen. Place the cursor on the Start button and click, the start menu will open, select **All Programs** and then another list will appear with the programs that are installed in your computer, look for **Macromedia**, then select **Macromedia Dreamweaver 8**, and the program will be opened.

From the Dreamweaver 8 shortcut on the desktop . You can double click to open Dreamweaver.

Either of the two above give the Macromedia Dreamweaver 8 home page (figure 1)

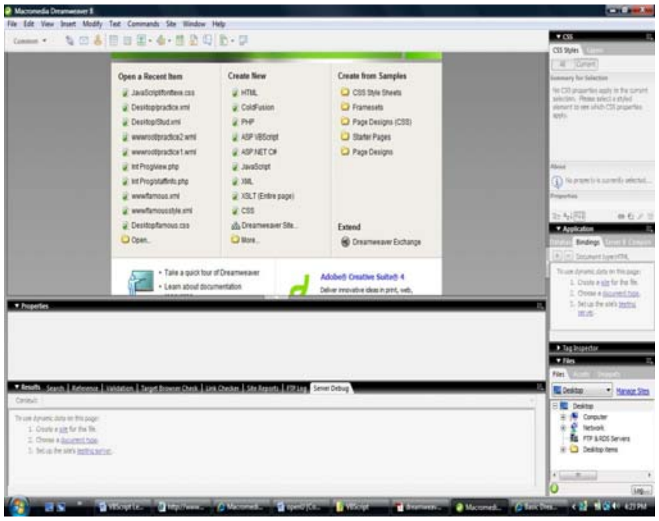


Fig.5.1: Dreamweaver Home Page

The start page enables you to open a recent document or create a new document. From the Start page you can also learn more about Dreamweaver by taking a product tour or a tutorial.

3.1 Opening and Saving Documents

3.1.1 Opening Document

Macromedia Dreamweaver front page consists of three captions: Open a Recent Item, Create New, and Create from Samples

Open a Recent Item: this option serve as shortcut to opening recent used files in Dreamweaver.

Create New: options here include HTML, PHP, XML, etc depending on the type of file you intend to create. Clicking on any option in "Create New" opens the workspace layout

The Workspace Layout

Dreamweaver provides an all-in-one-window integrated layout. In the integrated workspace, all windows and panels are integrated into a single larger application window.

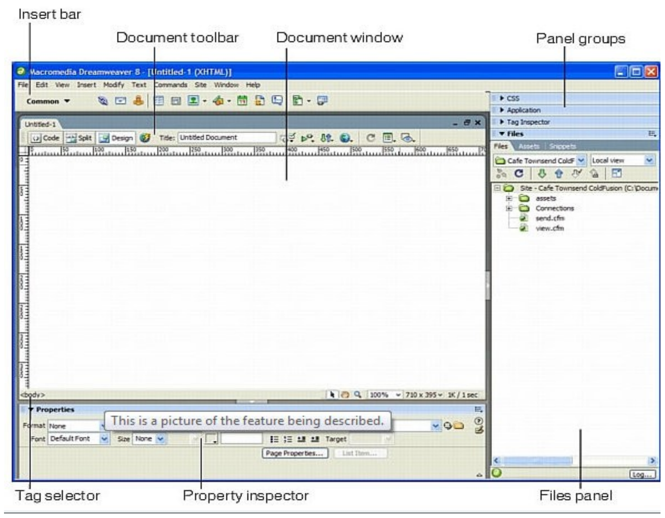


Fig.5.2: Dreamweaver Workspace

3.2 Dreamweaver Workspace Elements

This section briefly describes some elements of the Dreamweaver workspace.

NOTE

Dreamweaver provides many other panels, inspectors, and windows. To open Dreamweaver panels, inspectors, and windows, use the Window menu. If you can't find a panel, inspector, or window that's marked as open, select Window > Arrange Panels to neatly lay out all open panels.

The Insert bar contains buttons for inserting various types of "objects," such as images, tables, and layers, into a document. Each object is a piece of HTML code that enables you to set various attributes as you insert it. For example, you can insert a table by clicking the table button in the insert bar. If you prefer, you can insert objects using the insert menu instead of the insert bar.

The document toolbar contains buttons that provide options for different views of the document window (such as Design view and Code view), various viewing options, and some common operations such as previewing in a browser.

The standard toolbar (not displayed in the default workspace layout) contains buttons for common operations from the File and Edit menus: New, Open, Save, Save All, Cut, Copy, Paste, Undo, and Redo. To display the Standard toolbar, select View > Toolbars > Standard.

The coding toolbar (displayed in Code view only) contains buttons that let you perform many standard coding operations.

The Style Rendering toolbar (hidden by default) contains buttons that let you see how your design would look in different media types if you used media-dependent style sheets. It also contains a button that lets you enable or disable CSS styles.

The Document window displays the current document as you create and edit it.

The Property inspector lets you view and change a variety of properties for the selected object or text. Each kind of object has different properties. The property inspector is not expanded by default in the coder workspace layout.

The tag selector in the status bar at the bottom of the document window shows the hierarchy of tags surrounding the current selection. Click any tag in the hierarchy to select that tag and all its contents.

Panel groups are sets of related panels grouped together under one heading. To expand a panel group, click the expander arrow at the left of the group’s name; to undock a panel group, drag the gripper at the left edge of the group’s title bar.

The files panel enables you to manage your files and folders, whether they are part of a Dreamweaver site or on a remote server. The files panel also enables you to access all the files on your local disk, much like Windows Explorer (Windows) or the Finder (Macintosh).

3.3 The Elements of Insert Bar

The insert bar contains buttons for creating and inserting objects such as tables, layers, and images. When you roll the pointer over a button, a tooltip appears with the name of the button.

The insert bar is organised in the following categories:

a. The Common category enables you to create and insert the most commonly used objects, such as images and tables.
b. The Layout category enables you to insert tables, div tags, layers, and frames. You can also choose among three views of tables: Standard (default), Expanded Tables, and Layout. When Layout mode is selected, you can use the Dreamweaver layout tools: Draw Layout Cell and Draw

Layout Table

a. The Forms category contains buttons for creating forms and inserting form elements.
b. The Text category enables you to insert a variety of text- and list-formatting tags, such as b, em, p, h1, and u.
c. The HTML category enables you to insert HTML tags for horizontal rules, head content, tables, frames, and scripts.
d. Server-code categories are available only for pages that use a particular server language, including ASP, ASP.NET, CFML Basic, CFML Flow, CFML Advanced, JSP, and PHP. Each of these categories provides server-code objects that you can insert in Code view.
e. The Application category enables you to insert dynamic elements such as recordsets, repeated regions, and record insertion and update forms.
f. The Flash elements category enables you to insert Macromedia Flash elements.
g. The Favorites category enables you to group and organise the insert bar buttons you use the most in one common place.

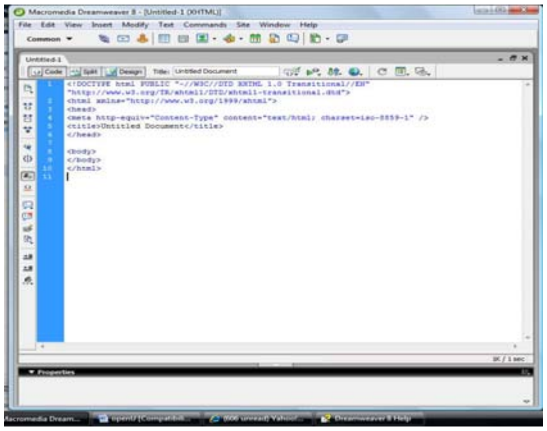
NOTE
The Windows workspace also has a Coder option, which docks the panel groups on the left side and displays the Document window in Code view by default.

3.4 The Document Window

This is the Dreamweaver work area. It two view: Design view and Code view.

Design view is a design environment for visual page layout, visual editing, and rapid application development. In this view, Dreamweaver displays a fully editable, visual representation of the document, similar to what you would see viewing the page in a browser.

Code view is a hand-coding environment for writing and editing HTML, JavaScript, server-language code--such as PHP or ColdFusion Markup Language (CFML)--and any other kind of code.


Macromedia Dreamweaver 8 - [Untitled-1 (XHTML)]
File Edit View Insert Modify Text Commands Site Window Help
Common
Untitled-1
Code Split Design Title: Untitled Document
Properties

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Untitled Document</title>
</head>

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

Fig.5.3: Dreamweaver Document Window

Dreamweaver provides the basic code for all application it supports. The programmer is only required to add to details to suit their needs.

3.5 Working with Dreamweaver

Here we will discuss how to accomplish basic tasks such as creating, opening, and saving files. Let start with files

Dreamweaver Files
You can work with a variety of file types in Dreamweaver. The primary kind of file you will work with is the HTML file. You can save HTML files with either the .html or .htm extension. The default is .html extension.

Other files include:

a. CSS, or Cascading Style Sheet files, have a .css extension. They are used to format HTML content and control the positioning of various page elements.
b. GIF, or Graphics Interchange Format files, have a .gif extension. The JPEG format is best for digital or scanned photographs, images using textures, images with gradient colour transitions, and any images that require more than 256 colours.

c. XML, or Extensible Markup Language files, have a .xml extension.
d. XSL, or Extensible Stylesheet Language files, have an .xsl or .xslt extension. They are used to style XML data that you want to display on a web page.
e. CFML, or ColdFusion Markup Language files, have a .cfm extension. They are used to process dynamic pages.
f. ASPX, or ASP.NET files, have an .aspx extension. They are used to process dynamic pages. PHP, or PHP: Hypertext Preprocessor files, have a .php extension. They are used to process dynamic pages.

3.6 Creating New Files in Dreamweaver

3.6.1 How to create a new blank document

1. Select File from the Menu Bar, click on New.
2. The New Document dialog box appears. The general tab is already selected.
3. From the category list, select Basic Page, Dynamic Page, Template Page, Other, or Framesets; then, from the list on the right, select the type of document you want to create.
4. For example, select Basic Page to create an HTML document, or select Dynamic page to create a ColdFusion or ASP document, and so on.
5. For more information about options in this dialog box, click the Help button in the dialog box.
6. Click the Create button.
7. The new document opens in the Document window.
8. Save the document.

3.7 Saving Files in Dreamweaver

To save a new document:

1. Select File from Menu bar, click on save.
2. TIP: In the dialog box that appears, navigate to the folder where you want to save the file.
3. In the File Name text box, type a name for the file.

NOTE: - It’s a good idea to save your file in a Dreamweaver site. Avoid using spaces and special characters in file and folder names and do not begin a filename with a numeral. In particular, do not use special characters (such as é, ç, or ¥) or punctuation (such as colons, slashes, or periods) in the names of files you intend to put on a remote server; many servers change these characters during upload, which will cause any links to the files to break.

Click Save.

3.8 Opening Files in Dreamweaver

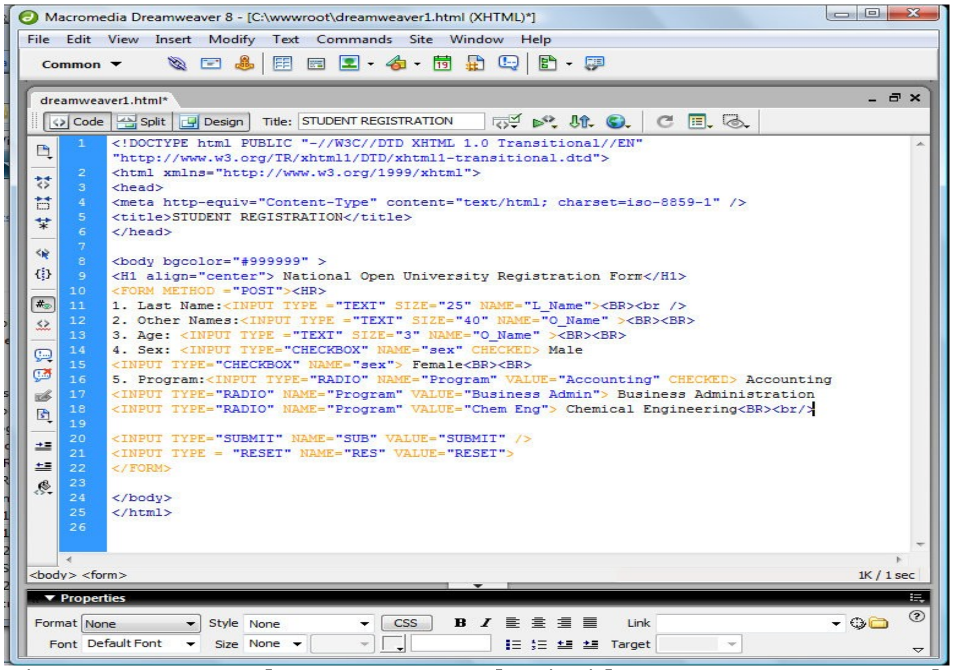
1. Dreamweaver allows you open and edit your documents.
2. To open a file:
3. Select File > Open.
4. In the Open dialog box, select the file and click open.
5. SELF-ASSESSMENT EXERCISE
6. List the step to follow to accomplish the following:
7. Creating new document
8. Opening files, and
9. Saving files in Dreamweaver.

Example 1: Creating a simple web page using Dreamweaver

Steps 1:
1. Go to start, click All Programs
2. Select Macromedia and click Macromedia Dreamweaver 8
3. Under Create New, select HTML. The resulting page looks like figure 3(Dreamweaver Document Window) above :
4. You will observe that the standard HTML tags such as the HTML, title, head and body tags have been written. Dreamweaver provides you the basic code. All you need to do is to add you're the required code that will give you what you intend to achieve.

Step 2:
1. To rename the file: go to file menu, click save as
2. On the save as dialog window, go to directories (Save in) navigate to My Computer, Local Disk (C:), then select wwwroot
3. At the file name, type Student.html
4. At the save as type: select All Documents
5. Click Save

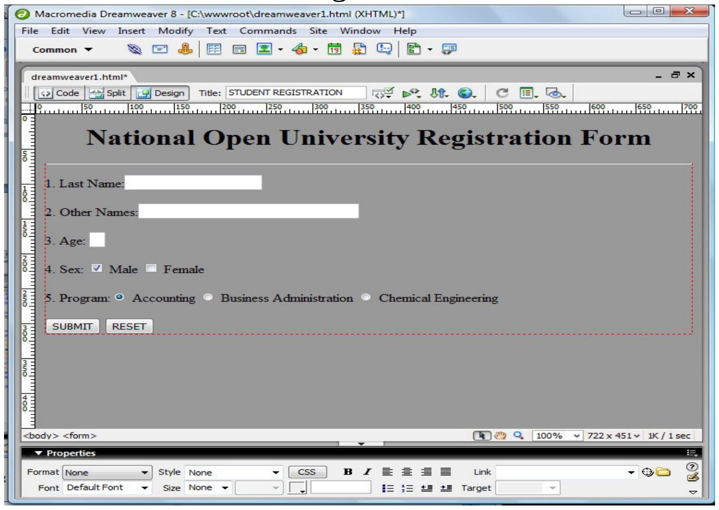
Step 3:
1. Now edit the code window to suit your purpose. Remember the Dreamweaver document window is divided into two: the code window (shown in figure 3) and the design window, which displays your web page as it will appear on normal web browsers.
2. Inside the body tag (<body>, </body>),



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>STUDENT REGISTRATION</title>
</head>

<body bgcolor="#999999" >
<H1 align="center"> National Open University Registration Form</H1>
<FORM METHOD ="POST"><HR>
1. Last Name:<INPUT TYPE ="TEXT" SIZE="25" NAME="L_Name"><BR><br />
2. Other Names:<INPUT TYPE ="TEXT" SIZE="40" NAME="O_Name" ><BR><BR>
3. Age: <INPUT TYPE ="TEXT" SIZE="3" NAME="O_Name" ><BR><BR>
4. Sex: <INPUT TYPE="CHECKBOX" NAME="sex" CHECKED> Male
<INPUT TYPE="CHECKBOX" NAME="sex"> Female<BR><BR>
5. Program:<INPUT TYPE="RADIO" NAME="Program" VALUE="Accounting" CHECKED> Accounting
<INPUT TYPE="RADIO" NAME="Program" VALUE="Business Admin"> Business Administration
<INPUT TYPE="RADIO" NAME="Program" VALUE="Chem Eng"> Chemical Engineering<BR><br/>

<INPUT TYPE="SUBMIT" NAME="SUB" VALUE="SUBMIT" />
<INPUT TYPE = "RESET" NAME="RES" VALUE="RESET">
</FORM>

</body>
</html>
```

Fig.5.4: Sample HTML Code inside Dreamweaver Code Window for <u>Practice 1</u>

Step 4: view the output of your work in the design window as shown in figure 5



National Open University Registration Form

1. Last Name:

2. Other Names:

3. Age:

4. Sex: ✓ Male ☐ Female

5. Program: ● Accounting ○ Business Administration ○ Chemical Engineering

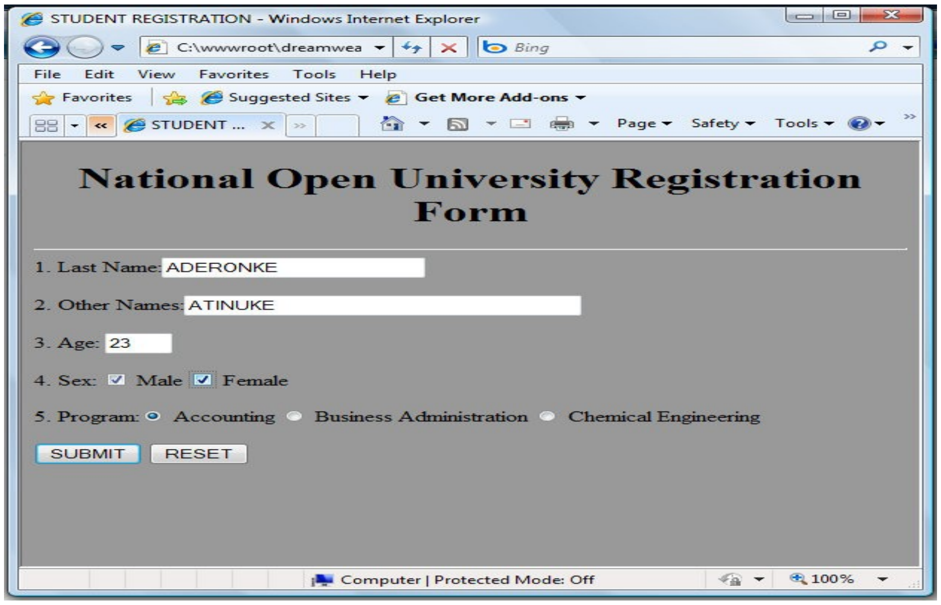
SUBMIT RESET

Fig.5.5: Dreamweaver Design Window for Example 1

Note: that any changes made to your program when it is on design view will alter the code window

Step 5: Run your program

Locate the wwwroot and open the Student.html file using Internet Explorer. Figure 7 shows the output.


STUDENT REGISTRATION - Windows Internet Explorer
C:\wwwroot\dreamwea
Bing
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Get More Add-ons
STUDENT ...
Page Safety Tools
SUBMIT
RESET
Computer | Protected Mode: Off
100%

National Open University Registration Form

1. Last Name: ADERONKE

2. Other Names: ATINUKE

3. Age: 23

4. Sex: [✓ Male] [✓ Female]

5. Program: [✓ Accounting] Business Administration Chemical Engineering

Fig.5.7: Browser Output of Example 1

SELF-ASSESSMENT EXERCISE

Create a simple web page of your choice using Dreamweaver by following the same procedures for Example 1 above.

4.0 CONCLUSION

Dreamweaver is a web page editor, for creating web pages which includes many new features that help you build websites and applications with a minimal amount of time and effort. Dreamweaver makes complex technologies simple and accessible, helping you accomplish more in less time. Additional features were added to Dreamweaver 8 from its predecessor Dreamweaver MX 2004.

5.0 SUMMARY

In this unit, you have learned how to get started with Dreamweaver, Opening and saving documents in Dreamweaver, Dreamweaver workspace elements, how to work with Dreamweaver, Creating new files in Dreamweaver and Opening and saving files in Dreamweaver.

6.0 TUTOR-MARKED ASSIGNMENT

Using Dreamweaver environment, develop a web site for an Online Gift Items Shop, displaying their available goods and prices.

7.0 REFERENCES/FURTHER READING

David, S. M. (2006). "Dreamweaver 8: The Missing Manual."

Janine, W. (2005). "Teach Yourself VISUALLY Macromedia Dreamweaver 8". Visual.