MCA(S6)21

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Housefed Complex, Dispur, Guwahati - 781 006

Master of Computer Applications

ADVANCED WEB TECHNOLOGY

CONTENTS

Subject Expert

Prof. Anjana Kakati Mahanta, Deptt. of Computer Science, Gauhati University
Prof. Jatindra Kr. Deka, Deptt. of Computer Science and Engineering,
Indian Institute of Technology Guwahati
Prof. Diganta Goswami, Deptt. of Computer Science and Engineering,
Indian Institute of Technology Guwahati

Course Coordinator

Tapashi Kashyap Das, Assistant Professor, Computer Science, KKHSOU

SLM Preparation Team

Units	Contributor
1 to 7	Nanu Alan Kachari Scientific Officer Grade II Department of Computer Science and Engineering, Indian Institute of Technology Guwahati

Dec 2013

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

The university acknowledges with thanks the financial support provided by the **Distance Education Council, New Delhi**, for the preparation of this study material.

Housefed Complex, Dispur, Guwahati- 781006; Web: www.kkhsou.net

COURSE INTRODUCTION

The Course on ***Advanced Web Technology*** aims to provide the reader, a brief overview of the history of the World Wide Web at the back drop to appreciate the current Web technologies available. While the Course is intended to expose the reader to some of the programming languages and Web Application Development concepts, it takes more of a hands-on approach. The reader is taken through the process of applying what was learnt in each Unit. The Units are progressive and builds on the previous Units covered, to eventually build a fully functioning Web Application using all the know-how gained along this course. All the code examples shown in the various Units of this course are tested and commented where ever neccessary for ease of code readability and understandability.

The *Unit 1* on Internet Concepts, aims to provide a heads up on the History of the Web and the various terminologies associated with it, including the underlying technologies, markup languages used. The *Unit 2* on HTML, tries to provide the reader the basic understanding of HTML with the do-it-with-examples approach. The *Unit 3* on CSS, covers the basic concepts of styling web pages and guides the reader into styling web pages. The *Unit 4* on JavaScript, provides a brief introduction to the JavaScript client-side scripting language and teaches the reader to apply and use JavaScript to the web pages created in the previous Units. *Unit 5* on XML and Ajax, provides a brief introduction to XML and Ajax and provides the reader with examples, the technique to use XML and Ajax on web pages. *Unit 6* on PHP, covers the PHP server-side scripting language with the goal to build a functional web applicaton. Finally, *Unit 7* covers the web application development using the MVC architecture. This unit teaches the reader to create an MVC model from scratch and build a functional web application using it.

Each unit of this course includes some along-side boxes to help you know some of the difficult, unseen terms. Some “EXERCISES” have been included to help you apply your own thoughts. You may find some boxes marked with: “LET US KNOW”. These boxes will provide you with some additional interesting and relevant information. Again, you will get “CHECK YOUR PROGRESS” questions. These have been designed to make you self-check your progress of study. It will be helpful for you if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with “ ANSWERS TO CHECK YOUR PROGRESS” given at the end of each unit.

MASTER OF COMPUTER APPLICATIONS
Advanced Web Technology
DETAILED SYLLABUS

Unit 1: Internet Concepts (Marks: 15)

History of the Internet and the Web. World Wide Web Consortium (W3C). Hypertext, hyperlink, Uniform Resource Locator (URL). Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), Domain Name System (DNS), TCP/IP the protocol of the Internet, Internet Protocol (IP) and the concept of IP Addresses. Internet Service Provider (ISP), Autonomous System (AS). Web Browser and Web Server. Free and Proprietory Software.

Unit 2: HTML (Marks: 15)

Hypertext Markup Language (HTML) and its components, HTML tags and attributes, Text formatting tags, List tags, Image tags, HTML tables, HTML Forms. Document Object Model (DOM), Applying what we have learnt I - Creating a simple web page, Applying what we have learnt II – Adding a Form to the web page.

Unit 3: CSS (Marks: 15)

Cascading Style Sheets (CSS) – Inline Style, Embedded Style, External Style Sheet, Imported Style Sheet, Ruleset, @ rule, Class Selector, ID Selector, Contextual Selector, Attribute Selector. CSS Properties – background properties, text properties, border properties. Applying what we have learnt III – Creating a CSS file, Applying what we have learnt IV – Using CSS in a web page.

Unit 4: JavaScript (Marks: 15)

JavaScript - Data types, Comparison Operators, Methematical Operators, Comments, document.write( ), console.log( ), Variables, length, substring, Conditional Statements - if, Loops - for, Functions. HTML DOM and JavaScript - Finding HTML Elements, Changing HTML elements, DOM events. Applying what we have learnt V – Creating a JavaScript file, Applying what we have learnt VI – Using JavaScript in a Web page.

Unit 5: XML and Ajax (Marks: 15)

XML - Declaration, Root Element, Child Elements, Element Attributes, Entity References, Comments. Ajax - XMLHttpRequest Object, Sending Ajax requests, Handling Ajax Responses. Applying what we have learnt VII – Adding Ajax Functionality in JavaScript. Applying what we have learnt VIIII – Adding Ajax Functionality to a Web Page.

Unit 6: PHP (Marks: 15)

Software Prerequisites - Installing Apache and PHP on Fedora 18, Starting and Testing Apache on Fedora 18, Testing PHP with phpinfo(), Installing MySQL on Fedora 18, Starting and Testing MySQL on Fedora 18, Installing the php-mysql Module, Checking the php-mysql Module. Getting Started with PHP - Basic PHP Syntax, Data Types, Variables, Constants, Operators, Control Structures, Functions. Applying what we have learnt IX – Connecting to MySQL using PHP. Applying what we have learnt X – Building a Web Page using PHP

Unit 7: Creating a Web Application – putting it all together (Marks: 10)

The MVC Design Pattern – Basic Web Architecture, MVC Architecture, Coding Considerations. Setting up our Development Environment. Building our MVC Framework. Building a PHP Application on our MVC framework.

MASTER OF COMPUTER APPLICATIONS
Advanced Web Technology
DETAILED SYLLABUS

Unit 1: Internet Concepts (Marks: 15)

History of the Internet and the Web. World Wide Web Consortium (W3C). Hypertext, hyperlink, Uniform Resource Locator (URL). Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), Domain Name System (DNS), TCP/IP the protocol of the Internet, Internet Protocol (IP) and the concept of IP Addresses. Internet Service Provider (ISP), Autonomous System (AS). Web Browser and Web Server. Free and Proprietory Software.

Unit 2: HTML (Marks: 15)

Hypertext Markup Language (HTML) and its components, HTML tags and attributes, Text formatting tags, List tags, Image tags, HTML tables, HTML Forms. Document Object Model (DOM), Applying what we have learnt I - Creating a simple web page, Applying what we have learnt II – Adding a Form to the web page.

Unit 3: CSS (Marks: 15)

Cascading Style Sheets (CSS) – Inline Style, Embedded Style, External Style Sheet, Imported Style Sheet, Ruleset, @ rule, Class Selector, ID Selector, Contextual Selector, Attribute Selector. CSS Properties – background properties, text properties, border properties. Applying what we have learnt III – Creating a CSS file, Applying what we have learnt IV – Using CSS in a web page.

Unit 4: JavaScript (Marks: 15)

JavaScript - Data types, Comparison Operators, Methematical Operators, Comments, document.write( ), console.log( ), Variables, length, substring, Conditional Statements - if, Loops - for, Functions. HTML DOM and JavaScript - Finding HTML Elements, Changing HTML elements, DOM events. Applying what we have learnt V – Creating a JavaScript file, Applying what we have learnt VI – Using JavaScript in a Web page.

Unit 5: XML and Ajax (Marks: 15)

XML - Declaration, Root Element, Child Elements, Element Attributes, Entity References, Comments. Ajax - XMLHttpRequest Object, Sending Ajax requests, Handling Ajax Responses. Applying what we have learnt VII – Adding Ajax Functionality in JavaScript. Applying what we have learnt VIIII – Adding Ajax Functionality to a Web Page.

Unit 6: PHP (Marks: 15)

Software Prerequisites - Installing Apache and PHP on Fedora 18, Starting and Testing Apache on Fedora 18, Testing PHP with phpinfo(), Installing MySQL on Fedora 18, Starting and Testing MySQL on Fedora 18, Installing the php-mysql Module, Checking the php-mysql Module. Getting Started with PHP - Basic PHP Syntax, Data Types, Variables, Constants, Operators, Control Structures, Functions. Applying what we have learnt IX – Connecting to MySQL using PHP. Applying what we have learnt X – Building a Web Page using PHP

Unit 7: Creating a Web Application – putting it all together (Marks: 10)

The MVC Design Pattern – Basic Web Architecture, MVC Architecture, Coding Considerations. Setting up our Development Environment. Building our MVC Framework. Building a PHP Application on our MVC framework.

UNIT 1: INTERNET AND WEB CONCEPTS

UNIT STRUCTURE

1.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the history of the Web
- learn about W3C, an international community who develops Web Standards
- learn the basic concepts of the Web
- learn about Free and Proprietary Software

1.2 INTRODUCTION

The Web or the World Wide Web (WWW) is a global information medium accessible via devices which are connected to the Internet. The Web is a service that operates over the Internet. The history of the Internet dates back significantly further than that of the World Wide Web. The Internet is often also referred to as a network of networks. The current generation is thoroughly engaging with the Internet using the services offered by the World Wide Web, mostly without being aware of the underlying technologies. As the evolution of new technologies allow for a total abstraction of the Internet and the Web from its underlying complexities. This unit aims to provide a basic understanding of the Internet from the standpoint of a Web Application Developer and introduces the reader to some of the basic concepts of the Internet and the Web.

1.3 HISTORY OF THE WORLD WIDE WEB

1980: Tim Berners-Lee a British scientist and an independent contractor at the European Organization for Nuclear Research (CERN), Switzerland, built ENQUIRE, as a personal database of people and software models, but also as a way to play with hypertext; each new page of information in ENQUIRE had to be linked to an existing page.

1984: Tim Berners-Lee returned to CERN, and considered its problems of information presentation: physicists from around the world needed to share data, and with no common machines and no common presentation software.

1989: Tim Berners-Lee wrote a proposal for "a large hypertext database with typed links", but it generated little interest. His boss, Mike Sendall, encouraged Berners-Lee to begin implementing his system on a newly acquired NeXT workstation. NeXT was founded in 1985 by Apple Computer co-founder Steve Jobs, after being forced out of Apple, along with a few of his co-workers. Berners-Lee considered several names, including *Information Mesh*, *The Information*

Mine (turned down as it abbreviates to TIM, the WWW's creator's name) or *Mine of Information* (turned down because it abbreviates to MOI which is "Me" in French), but settled on *World Wide Web*.

1990: Tim Berners-Lee found an enthusiastic collaborator in Robert Cailliau, who rewrote the proposal (published on November 12, 1990) and sought resources within CERN. Berners-Lee and Cailliau pitched their ideas to the European Conference on Hypertext Technology in September 1990, but found no vendors who could appreciate their vision of marrying hypertext with the Internet. By Christmas of 1990, Berners-Lee had built all the tools necessary for a working Web: the HyperText Transfer Protocol (HTTP) 0.9, the HyperText Markup Language (HTML), the first Web browser (named WorldWideWeb, which was also a Web editor), the first HTTP server software (later known as CERN httpd), the first web server (http://info.cern.ch), and the first Web pages that described the project itself. The browser could access Usenet newsgroups and FTP files as well. However, it could run only on the NeXT; Nicola Pellow therefore created a simple text browser that could run on almost any computer called the Line Mode Browser. To encourage use within CERN, Bernd Pollermann put the CERN telephone directory on the web — previously users had to log onto the mainframe in order to look up phone numbers.

1991: In January 1991 the first Web servers outside CERN itself were switched on.

1992-1995: The early adopters of the World Wide Web were primarily university-based scientific departments or physics laboratories such as Fermilab and SLAC. By January 1993 there were fifty Web servers across the world; by October 1993 there were over five hundred. Early websites intermingled links for both the HTTP web protocol and the then-popular Gopher protocol, which provided access to content through hypertext menus presented as a file system rather than through HTML files. Early Web users would navigate either by bookmarking popular directory pages, such as Berners-Lee's first site at http://info.cern.ch/ which is still available, or by consulting updated lists such as the NCSA (National Center for Supercomputing Applications) "What's New" page. Some sites were also indexed

by WAIS (wide area information server), enabling users to submit full-text searches similar to the capability later provided by search engines.

There was still no graphical browser available for computers besides the NeXT. This gap was discussed in January 1992, and filled in April 1992 with the release of Erwise, an application developed at the Helsinki University of Technology, and later by ViolaWWW, created by Pei-Yuan Wei, which included advanced features such as embedded graphics, scripting, and animation. ViolaWWW was originally an application for HyperCard, an application program and programming tool for Apple Macintosh and Apple IIGS computers. Both programs ran on the X Window System for Unix. Students at the University of Kansas adapted an existing text-only hypertext browser, Lynx, to access the web. Lynx was available on both Unix and DOS.

The turning point for the World Wide Web was the introduction of the Mosaic web browser in 1993, a graphical browser developed by a team at the NCSA at the University of Illinois at Urbana-Champaign (UIUC), led by Marc Andreessen. Funding for Mosaic came from the High-Performance Computing and Communications Initiative, a funding program initiated by the then-Senator Al Gore's High Performance Computing and Communication Act of 1991 also known as the Gore Bill.

The first Microsoft Windows browser was Cello, written by Thomas R. Bruce for the Legal Information Institute at Cornell Law School to provide legal information, since more lawyers had more access to Windows than to Unix. Cello was released in June 1993. The NCSA released Mac Mosaic and WinMosaic in August 1993.

After graduation from UIUC, Andreessen and James H. Clark, former CEO of Silicon Graphics, met and formed Mosaic Communications Corporation to develop the Mosaic browser commercially. The company changed its name to Netscape in April 1994, and the browser was developed further as Netscape Navigator.

In May 1994, the first International WWW Conference, organized by Robert Cailliau, was held at CERN; the conference has been held every year since. In April 1993, CERN had agreed that anyone could use the Web protocol and code royalty-free; this was in part a reaction to the perturbation caused by the University of Minnesota's announcement that it would begin charging license fees for its implementation of the Gopher protocol.

In September 1994, Tim Berners-Lee founded the World Wide Web Consortium (W3C) at the Massachusetts Institute of Technology with support from the Defense Advanced Research Projects Agency (DARPA) and the European Commission. It comprised of various companies that were willing to create standards and recommendations to improve the quality of the Web. Berners-Lee made the Web available freely, with no patent and no royalties due. The W3C decided that its standards must be based on royalty-free technology, so they can be easily adopted by anyone.

1996-2001: Web becomes commercialized and this period saw the dot-com boom.

2002 onwards: During this period, a handful of companies found success developing business models that helped make the World Wide Web a more compelling experience. These include airline booking sites, Google's search engine and its profitable approach to simplified, keyword-based advertising, as well as ebay's do-it-yourself auction site and Amazon.com's online department store. This new era also begot social networking websites, such as MySpace and Facebook, which, though unpopular at first, very rapidly gained acceptance in becoming a major part of youth culture.

New ideas for sharing and exchanging content ad-hoc, such as Weblogs and RSS, rapidly gained acceptance on the Web. This new model for information exchange, primarily featuring DIY (do-it-yourself) user-edited and generated websites (like wikipedia, youtube. etc.), was coined Web 2.0. As Internet connectivity becomes ubiquitous, manufacturers have started to leverage the expanded computing power of their devices to enhance their usability and

capability. Through Internet connectivity, manufacturers are now able to interact with the devices they have sold and shipped to their customers, and customers are able to interact with the manufacturer (and other providers) to access new content.

1.4 WORLD WIDE WEB CONSORTIUM (W3C)

The World Wide Web Consortium (W3C) is the main international standards organization for the World Wide Web (abbreviated WWW or W3). Founded and currently led by Tim Berners-Lee, the consortium is made up of member organizations which maintain full-time staff for the purpose of working together in the development of standards for the World Wide Web. The W3C also engages in education and outreach, develops software and serves as an open forum for discussion about the Web. More information on W3C can be found at their website http://www.w3.org/

1.5 BASIC CONCEPTS

In this section, we will briefly discuss some of the terms and concepts of the Web that you may already be familiar with for understanding the World Wide Web.

1.5.1 Hypertext, Hyperlink, Uniform Resource Locator(URL)

Hypertext is the text displayed on a computer display or other electronic device with references (or hyperlinks) to other text. Hypertext is text with hyperlinks. Apart from text, hypertext is sometimes used to describe tables, images and other presentation content forms with hyperlinks. Hypertext is the underlying concept defining the structure of the World Wide Web, with pages often written in the Hypertext Markup Language (HTML).

Hyperlink (or link) is a reference to data. A hyperlink points to a whole document

or to a specific element within a document. Hyperlinks are typically activated by a mouse click, keypress sequence or by touching the screen.

Uniform resource locator (URL) is a specific character string that constitutes a reference to a resource. In most web browsers, the URL of a web page is displayed on top inside an address bar. URLs are commonly used for web pages (http://www.google.co.in), but can also be used for file transfer (ftp:), email (mailto:), telephone numbers (tel:) and many other applications.

1.5.2 Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is an application protocol and is the protocol to exchange or transfer hypertext. HTTP is the foundation of data communication for the World Wide Web. The standards development of HTTP was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) which led to the publication of a series of Requests for Comments (RFCs), most notably RFC 2616 (on June 1999), which defines the HTTP/1.1, the version of HTTP in common use.

HTTP is a stateless application layer protocol designed within the framework of the TCP/IP Suite. A HTTP session is a sequence of network request-response transactions. A HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server (port 80 is default though other ports can be used). A HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own. The body of this message is typically the requested resource, although an error message or other information may also be returned. HTTP provides a standardized way for computers to communicate with each other on the Internet. HTTP specification specifies how clients request data will be constructed and sent to the server, and how servers will respond to these requests.

1.5.3 Hypertext Transfer Protocol Secure (HTTPS)

The Hypertext Transfer Protocol Secure (HTTPS) is a secure method of accessing or sending information across the Internet. All data sent over HTTPS is encrypted before it is sent, this prevents anyone from viewing the information sent over the Internet, if intercepted. Because the data is encrypted over HTTPS, it is slower than HTTP, which is why HTTPS is used when requiring secure transfers. For example login information or with pages that contain sensitive information such as online banking. HTTPS uses port 443.

1.5.4 Domain Name System (DNS)

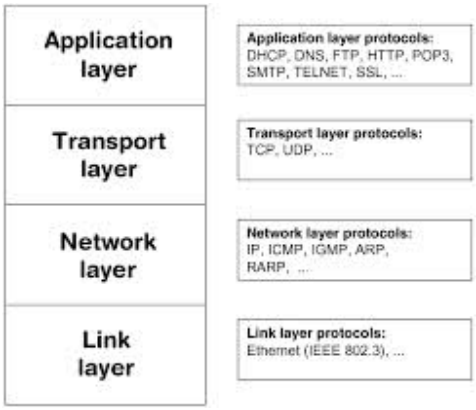
Domain Name System or DNS is a service which provides resolution of fully qualified domain names (FQDN) into IP addresses and vice-versa. What this means is that domain names like say “www.kkhsou.in” is easier to remember than some IP address like “182.50.130.66”. Moreover, IP addresses may change. Just think how difficult it would be to keep remembering numerical addresses.

DNS servers are also known as a nameservers as they provide a network service that associates hostnames with their respective IP addresses. DNS is usually implemented using one or more centralized servers that are authoritative for certain domains. When a client host requests information from a nameserver, it usually connects to port 53. The nameserver then attempts to resolve the name requested. If it does not have an authoritative answer, or does not already have the answer cached from an earlier query, it queries other nameservers, called root nameservers, to determine which nameservers are authoritative for the name in question, and then queries them to get the requested name.

1.5.5 TCP/IP - The Protocol of the Internet

The Internet protocol suite is the networking model and a set of communications protocols used for the Internet. It is commonly known as TCP/IP,

because its most important protocols, the Transmission Control Protocol (TCP) and the Internet Protocol (IP), were the first networking protocols defined in this standard. It is occasionally also known as the DoD (Department of Defence) model, because the development of the networking model was funded by DARPA, an agency of the United States Department of Defense.



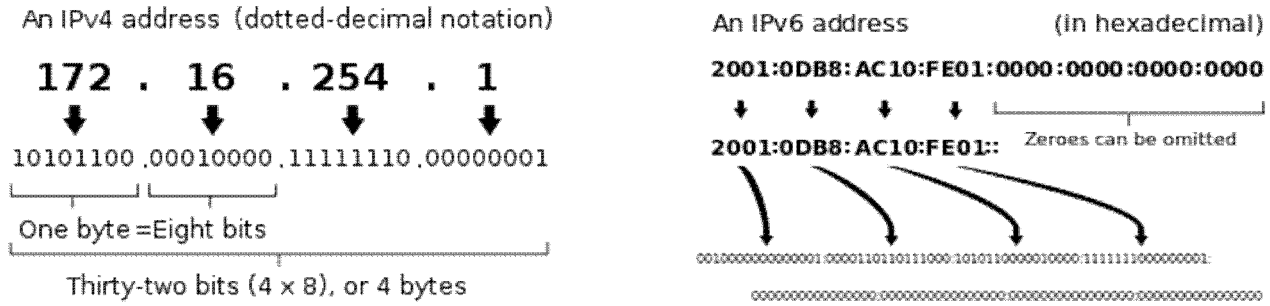
TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. This functionality has been organized into four abstraction layers. From lowest to highest, the layers are the *Link Layer*, containing communication technologies for a single network segment (link), the *Internet Layer*, connecting independent networks, thus establishing internetworking, the *Transport Layer* handling process-to-process communication, and the *Application Layer*, which interfaces to the user and provides support services. The TCP/IP model and related protocols are maintained by the Internet Engineering Task Force (IETF).

1.5.6 Internet Protocol (IP) and the Concept of IP Addresses

A Internet Protocol address is a numerical label assigned to each device (e.g., Laptop, Desktop, Server, Printer, Wifi Router, etc.) participating in a computer network that uses the Internet Protocol for communication. Two versions of the Internet Protocol are in use today, IPv4 (with 32-bit addresses) and IPv6 (with 128-bit addresses). IPv6 was developed in 1995 due to the enormous growth of the Internet and the predicted depletion of available IPv4 addresses.

IP Address: IP addresses are binary numbers, but they are usually stored

in text files and displayed in human-readable notations, such as 192.168.1.1 (for IPv4 in dotted decimal format i.e., 4 octets separated by periods, 4 sets of “numbers” 0 thru 255), and fda8:06c3:ce53:a890:0000:0000:0000:0001 (for IPv6 in hexadecimals, a double-octet format separated by a colon, 8 sets of “numbers” 0 thru 9, and a thru f).



[Picture Source: http://en.wikipedia.org]

Subnet Mask: A subnet is a logical subdivision of an IP network. The practice of dividing a network into two or more networks is called subnetting. A subnet mask determines which part of an IP address is the Network part and the Host part. In other words, a subnet mask specifies the number of bits used for identifying the network and hosts in an IP address.

In a computer network, the IP Address used by each device has to be a unique one for a successful communication between the devices.

The Internet Assigned Numbers Authority (IANA) manages the IP address space allocations globally and delegates five regional Internet registries (RIRs) to allocate IP address blocks to local Internet registries (Internet service providers) and other entities.

1.5.7 Internet Service Provider (ISP), Autonomous System (AS)

An Internet Service Provider (ISP) is the gateway to the Internet though an ISP can also provide services, and not limited to, Internet access, Internet

transit, domain name registration and hosting, email, dial-up access, leased line access and colocation.

An Autonomous System (AS) is a connected group of one or more IP prefixes (network prefix) run by one or more network operators which has a single and clearly defined routing policy. An ISP must have an officially registered Autonomous System Number (ASN). A unique ASN is allocated to each AS for use in BGP routing. AS numbers are important because the ASN uniquely identifies each network on the Internet.

1.5.8 Web Browser and Web Server

A web server software helps to deliver web content that can be accessed via the Internet. Some web server softwares include Apache, Microsoft IIS, nginx, Google GWS, etc.

A web browser is a software application for retrieving, presenting and traversing information resources on the World Wide Web. An information resource is identified by a Uniform Resource Identifier (URI/URL) and may be a web page, image, video or other piece of content. Hyperlinks present in resources enable users easily to navigate their browsers to related resources. Some web browsers include the Google Chrome, Mozilla Firefox, Internet Explorer, Opera, Safari, etc.

1.6 FREE AND PROPRIETARY SOFTWARE

Free software is a computer software that anyone is freely licensed to use, copy, study, and change the software in any way, and the source code is openly shared so that people are encouraged to voluntarily improve the design of the software.

Proprietary software is a computer software licensed under exclusive legal right of the copyright holder with the intent that the licensee is given the right to use the software only under certain conditions, and restricted from other uses, such as modification, sharing, studying, redistribution, or reverse

engineering. Usually the source code of proprietary software is not made available.

1.7 LET US SUM UP

In this unit, we have tried to get acquainted briefly with the World Wide Web. Though we have covered only a few of the topics, these topics are intended to inspire and point you in the direction to further explore. This unit was not intended to provide you with an exhaustive in-depth on the topic but merely to introduce you to some of the basic concepts.

- What we have learned in this unit.
- Brief history of the World Wide Web.
- W2C.
- Hypertext, Hyperlink, URI/URL, HTTP/HTTPS.
- TCP/IP, DNS, ISP, Autonomous System.
- Web Browser, Web Server.
- Free and Proprietary Software.

CHECK YOUR PROGRESS

Q1: Which of the following rightly describes the Internet?

(a) Intranet (b) LAN (c) WAN (d) Network of Networks

Q2: Which of the following is an expansion of the abbreviation WWW?

(a) World Wide Wireless (b) World Wide Web

(c) Web World Wireless (d) Wireless World Web

Q3: Which of the following is true with regards to WWW?

(a) Internet operates over WWW. (b) WWW operates over Internet.

(c) WWW and Internet both operate over WWW.

(d) None of the options (a), (b), or (c) are true.

Q4: What does W3C stand for and who is attributed to founding the W3C?

Q5: Which of the following periods saw the so called *dot-com* boom?

(a) 1969-1971 (b) 1980-1990 (c) 1991-1996 (d) 1996-2001

Q6: Which of the following statements best describes Hyperlink and Hypertext?

(a) Hyperlink is reference to data and Hypertext is text with Hyperlinks.

(b) Hypertext is reference to data and Hyperlink is text with Hypertexts.

(c) Hyperlink is reference to Hypertext and Hypertext is text with Hyperlinks.

(d) Hypertext is reference to Hyperlink and Hyperlink is text with Hypertexts.

Q7: Which of the following are examples of a URL?

(a) ftp://ftp.abc.com/abc.txt (b) http://www.abc.com/abc.txt

(c) mailto:abcd@wxyz.com (d) tel:+91-361-123-4567

Q8: Which of the following statements are true?

(a) HTTP is the application layer protocol used to transfer Hypertext.

(b) HTTP is a stateless protocol.

(c) HTTPS is used for secure online transactions.

(d) HTTPS is slower compared to HTTP.

Q9: Which of the following statements are true for DNS?

(a) DNS stands for Domain Name System.

(b) DNS provides FQDN resolution to IP Addresses and vice-versa.

(c) DNS service uses the TCP port 443 by default.

(d) All the above options (a), (b), (c).

Q10: Which of the following statements are true?

(a) TCP/IP has four layers viz. Application, Transport, Network and Link Layers.

(b) IP Addresses must be unique in a TCP/IP Network.

(c) The Internet Assigned Numbers Authority (IANA) manages IP Address allocations.

(d) Free Software gives you the freedom to use, copy, study, and change the software/source code (which is licensed under the Free Software).

1.8 FURTHER READINGS

- Andrew. S.Tanenbaum and David J. Wetherall, Computer Networks 5th Edition, Pearson, ISBN-13 9789332518742, ISBN-10 9332518742, September 2013

- Larry L. Peterson, Computer Networks : A System Approach 5th Edition, Morgan Kaufmann Publishers, ISBN-13 9789380501932, ISBN-10 9380501935, 2011

- http://en.wikipedia.org/wiki/History_of_the_World_Wide_Web

- http://www.w3.org/

- http://en.wikipedia.org/wiki/Hypertext

- http://en.wikipedia.org/wiki/Hyperlink

- http://en.wikipedia.org/wiki/Uniform_resource_identifier

- http://tools.ietf.org/html/rfc2616

- http://tools.ietf.org/html/rfc5246

- http://tools.ietf.org/html/rfc1122

- http://tools.ietf.org/html/rfc2181

- http://en.wikipedia.org/wiki/Internet_service_provider

- http://tools.ietf.org/html/rfc4271

- http://en.wikipedia.org/wiki/History_of_the_web_browser

- http://www.gnu.org/philosophy/categories.html

1.9 ANSWERS TO CHECK YOUR PROGRESS

1. (d)

2. (b)

3. (b)

4. W3C stands for the World Wide Web Consortium. Tim Berners-Lee founded the W3C in September 1994.

5. (d)

6. (a)

7. (a), (b), (c), (d) all are examples of a URL.

8. (a), (b), (c), (d) all are true.

9. (a), (b) are true.

10. (a), (b), (c), (d) all are true.

1.10 MODEL QUESTIONS

1. What is the difference between the Internet and the Web?
2. Who founded the W3C and when?
3. What is W3C?
4. What is a Hyperlink?
5. What is the difference between HTTP and HTTPS?
6. What is the purpose of DNS?
7. Which communication protocol is most commonly used on the Internet?
8. How many bits is an IP Address for IPv4 Addresses?
9. What is IP Address subnetting?
10. What is Free Software?

******

UNIT 2: HTML

UNIT STRUCTURE

2.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the basics of HTML
- learn about DOM
- create your own HTML documents

2.2 INTRODUCTION

The Hypertext Markup Language (HTML) is a language for describing web pages, and aptly so, HTML documents are called web pages. It is a markup language meaning that is has a set of markup tags which describes the document content. HTML documents can contain HTML tags as well as plain text.. This unit aims to provide a basic understanding of HTML and its components for creating web pages.

2.3 HYPERTEXT MARKUP LANGUAGE (HTML) AND ITS COMPONENTS

2.3.1 HTML tags, Elements and Attributes

HTML tags are keywords or tag names surrounded by angle brackets like "<html>" and "</html>". HTML tags normally are in pairs like " " and " ". The first tag, like " ", in a pair is the start or opening tag and the second tag, like " ", is the end or closing tag. The end tag is written like the start tag, with a forward slash "/" before the tag name between angle brackets “< >”.

Some commonly used HTML tags are listed below.

<!-- ... **-->** : Comment Tags. *Example:* <!-- this is a comment -->

<html></html> : This tag tells the browser that the document is an HTML document. Also, the "<html>" tag represents the root of an HTML document containing other HTML elements. Furthermore, the “</html>” end tag should be included at the end of every HTML document.

<head></head> : The "<head>" tag is a container for all the header elements. The "<head>" tag can include the title for the document (eg.: <title>Hello World!</title>), scripts (eg.: <script>alert("Hello World!")</script>), styles (eg.: <style>p {color:blue;}</style>), meta information (eg.: <meta charset="UTF-8">), links (eg.: <link rel="stylesheet" type="text/css" href="mytheme.css">), etc. The "<head>" tag must also be closed using the "</head>" closing tag.

<body></body> : The "<body>" tag defines the HTML document body. The "<body>" tag contains all the contents of an HTML document, such as text, hyperlinks, images, tables, lists, etc. The "<body>" tag must also be closed using the "</body>" closing tag.

<noscript></noscript> : This tag defines an alternate content for users that have disabled scripts in their browser or have a browser that does not support script. The content inside the "<noscript>" tags will be displayed if scripts are not supported, or are disabled in the browser.
Example: **<noscript>**Script support disabled/not supported.**</noscript>**

HTML elements start with a start tag and end with an end tag. The element content is everything between the start and the end tags. Some HTML elements may have empty content and are closed in the start tag (eg.:
).
Example: ** Welcome to our Webpage **

HTML attributes are associated with HTML elements and provide additional information about an element. HTML attributes are always specified in the start tag and are specified as name="value" pairs.
Example: Click <a **href="http://www.kkhsou.in"**>here to go to KKHSOU

2.3.2 Text Formatting Tags

Some common text formatting HTML tags.

** ** : Defines a paragraph.
**** : Makes the text typed between these tags bold.

**** : Makes the text typed between these tags italic.
<u></u> : Makes the text typed between these tags underlined.
<code></code> : Makes the text appear like source code.
**** : Makes the text appear as subscript.
**** : Makes the text appear as superscript.

2.3.3 List Tags

There are basically three types of HTML list tags that are commonly in use. They are ordered lists “
”, unordered lists “
” and description lists “<dl></dl>”.

**
** : The ordered list starts with the "
" tag and ends with the "
" tag. Each list item in an ordered list, starts with the "
" tag and ends with the "
" tag. The list items in an ordered list are marked with numbers.
Example:

```
<ol>
        <li>First Item</li>
        <li>Second Item</li>
</ol>
```

**
** : The unordered list starts with the "
" tag and ends with the "
" tag. Each list item in an unordered list also starts with the "
" tag and ends with the "
" tag. However, the list items in an unordered list are marked with bullets instead.
Example:

```
<ul>
        <li>First Item</li>
        <li>Second Item</li>
</ul>
```

<dl></dl> : The description list is a list of terms and descriptions pairs. This list starts with the “<dl>” tag and ends with the “</dl>” tag. Each list item in the description list consists of a pair of “<dt></dt>” and “<dd></dd>” tags. The

“<dt></dt>” tags specifies the term name and the “<dd></dd>” tags specifies the description of the term.

Example:

```
<dl>
        <dt>Free Software</dt>
        <dd>Free software guarantees everyone equal rights to their programs; any user can study the source code of a free program, modify it and share it.</dd>
</dl>
```

2.3.4 Image Tags

We use the “<img>” tag in HTML to define images in web pages. The “<img>” tag is empty meaning that it contains attributes only, and does not have or need a closing tag.

To display an image on a page, you need to use the "src" attribute which specifies the image "source". The value of the “src” attribute is the URL of the image you want to display. Apart from the "src" attribute there are other attributes one could use.

Example:

```
<img src=”http://www.kkhsou.in/main/Banner/top.jpg”>
```

The "alt" attribute specifies an alternate text for an image, in case the image cannot be displayed.

Example:

```
<img src=”http://www.kkhsou.in/main/Banner/top.jpg” alt=”KKHSOU Banner”>
```

The "height" and "width" attributes can also be used to specify the height and width of an image in pixels.

Example:

```
<img src=”http://www.kkhsou.in/main/Banner/top.jpg” alt=”KKHSOU Banner” height="115" width="942">
```

2.3.5 HTML Tables

In HTML, tables are defined with the "<table></table>" tags. A table is divided into rows with the "<tr></tr>" tags and each row is divided into data cells with the "<td></td>" tags. A single row can also be specified as a heading with the "<th></th>" tags.

Example:

```
<table>
	<th>
		<td>Sl. No.</td><td>Roll No.</td><td>Student Name</td>
	</th>
	<tr>
		<td>1</td><td>1000001</td><td>ABCD EFGH</td>
	</tr>
	<tr>
		<td>2</td><td>1000002</td><td>IJKL MNOP</td>
	</tr>
	<tr>
		<td>3</td><td>1000003</td><td>QRST UVWX</td>
	</tr>
</table>
```

A HTML table can be specified with the following commonly used attributes as well.

The "border" attribute to specify the table border, "cellpadding" attribute to specify the space between the cell contents and its borders, "cellspacing" attribute to specify the space between the cells, with values in pixels.

Example:

```
<table border=”1” cellpadding=”5” cellpadding=”5”>
	<th>
		<td>Sl. No.</td><td>Roll No.</td><td>Student Name</td>
	</th>
	<tr>
		<td>1</td><td>1000001</td><td>ABCD EFGH</td>
```

```
        </tr>
        <tr>
                <td>2</td><td>1000002</td><td>IJKL MNOP</td>
        </tr>
        <tr>
                <td>3</td><td>1000003</td><td>QRST UVWX</td>
        </tr>
</table>
```

2.3.6 HTML Forms

When we need to pass data to a Web server from a client Web browser, we can use HTML Forms. A HTML form generally contains input elements like text fields, drop-down boxes, checkboxes, radio-buttons, submit buttons, etc. Though a HTML form can also contain select lists, textarea, fieldset, legend, and label elements. The "<form></form>" tags are used to create an HTML form.
Example:

```
<form name="login" action=”process.php” method=”post”>

        Username: <input type="text" name="username"><br />

        Password: <input type="password" name="password">

        <input type="submit" value="Login">

</form>
```

In the above example for the HTML forms, the form attribute “name” specifies the name of the form, “action” attribute specifies where the form data is sent when the form is submitted by pressing the submit button and the “method” attribute specifies the HTTP method the use (GET or POST) while sending the form data.

2.4 DOCUMENT OBJECT MODEL (DOM)

The **Document Object Model** is a platform and language neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. The Document Object Model (DOM) can be used in representing and interacting with objects in HTML. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its application programming interface (API).

When a HTML document is loaded into a web browser, it becomes a **HTML DOM document object**. The document object is the root node of the HTML document containing the element nodes, text nodes, attribute nodes, and comment nodes. The document object provides properties and methods to access all node objects, from within JavaScript. The document is a part of the window object and can be accessed as “window.document”.

The **HTML DOM element object** represents an HTML element. Element objects can have child nodes of type element nodes, text nodes, or comment nodes.

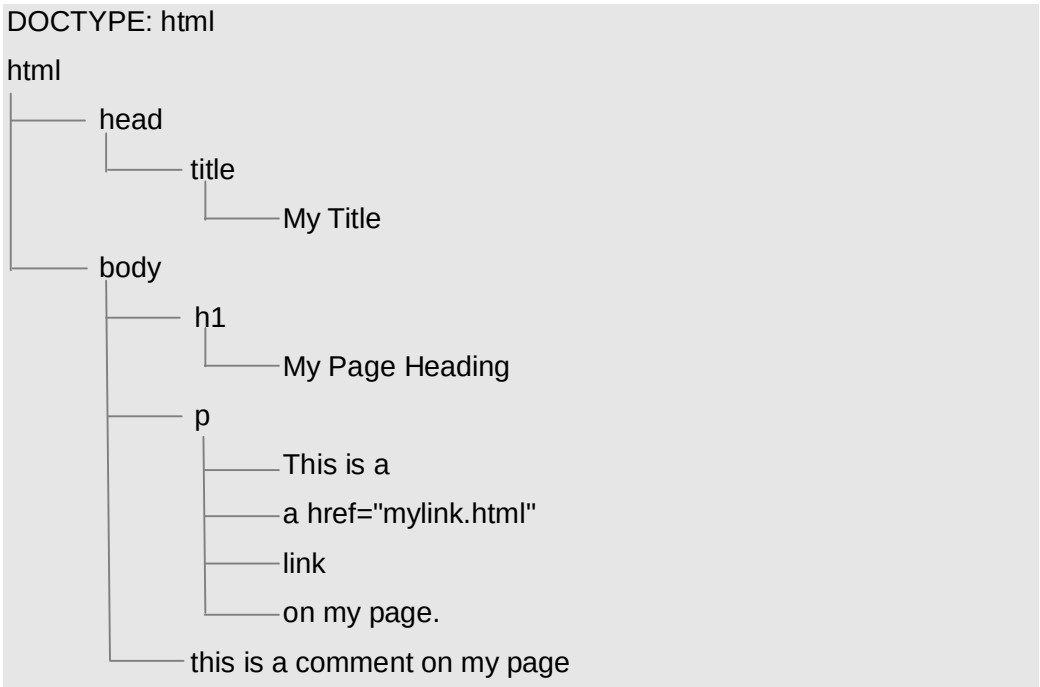
The **HTML DOM attribute object** represents an HTML attribute. An HTML attribute always belongs to an HTML element.

The **HTML DOM events** allow JavaScript to register different event handlers on elements in an HTML document. Events are normally used in combination with functions, and the function will not be executed before the event occurs, such as when a user clicks a button or presses a key on the keyboard.

Example: An HTML Document

```
<!DOCTYPE html>
<html>
<head>
        <title>My Title</title>
</head>
<body>
        <h1>My Page Heading</h1>
        <p>This is a <a href="mylink.html">link</a> on my page.</p>
        <!-- this is a comment on my page -->
</body>
</html>
```

Example: DOM Tree for the HTML Document above



We will revisit the DOM in Unit 4, with JavaScripts where the examples will help us understand what we have just briefly discussed in the paragraphs above.

2.5 APPLYING WHAT WE HAVE LEARNT I - CREATING A SIMPLE WEB PAGE

To start creating web pages or HTML documents, all we really need is a text editor. Though you have a plethora of text editor choices to choose from.

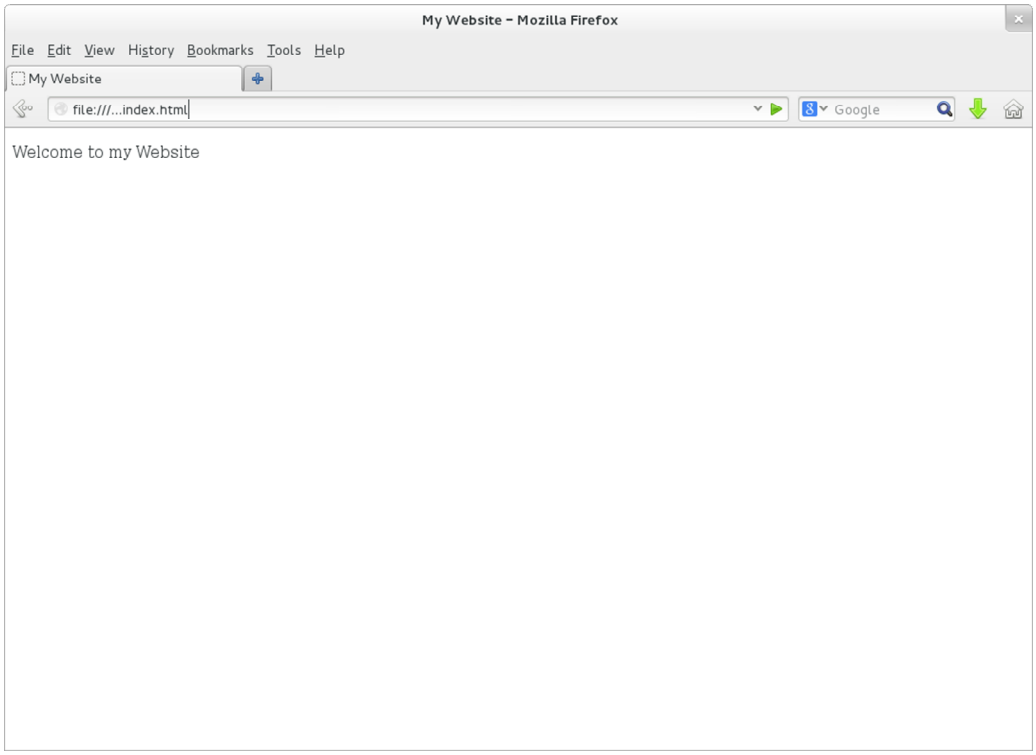
Let us start, by creating a HTML document named “index.html” with the contents as show below. Notice that the file extension we are using is “.html”. This HTML document is named “index.html” as we will use this document to be our index page. Don't worry if you do not understand this yet. We will cover all the concepts, wherever needed, as we progress through the Units in this course.

index.html

```
<!DOCTYPE html>
<html>
	<head>
		<title>My Website</title>
	</head>
	<body>
		<p>Welcome to my Website</p>
	</body>
</html>
```

Now, if we open this file in a web browser like Mozilla Firefox, Google Chrome, Microsoft Internet Explorer, etc., you will see a similar page on your browser as show in the figure below.

Index.html in a browser



2.6 APPLYING WHAT WE HAVE LEARNT II – ADDING A FORM TO THE WEB PAGE

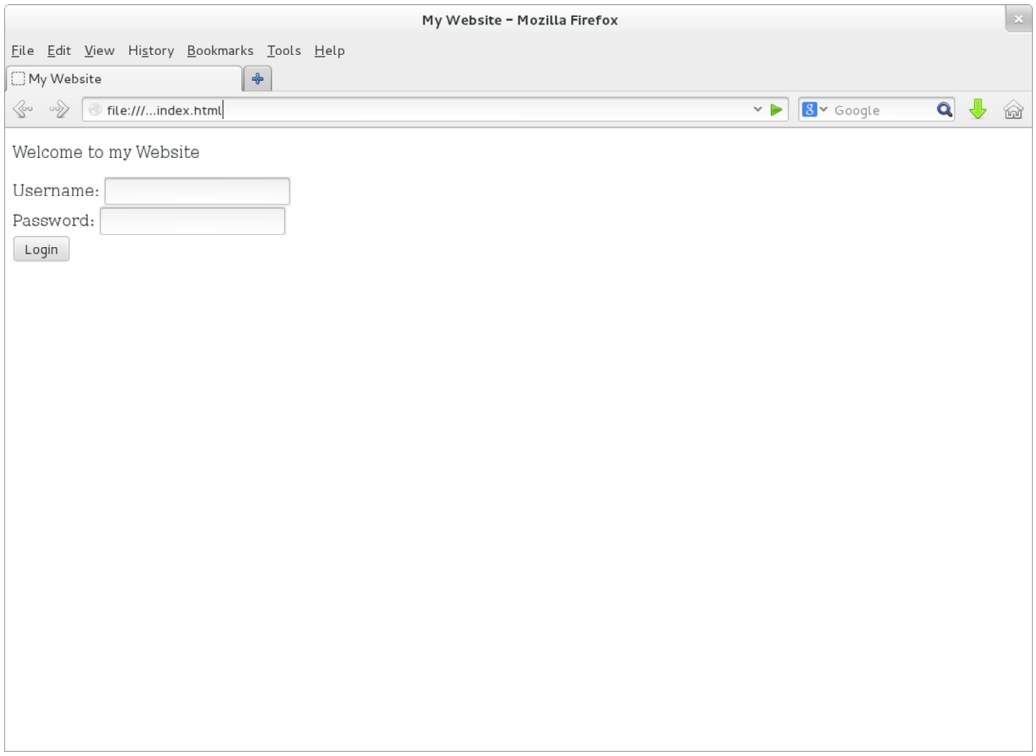
Now, we add a form to our web page and just to make it useful let us add a login form. The “index.html” file is modified with the content shown below.

index.html

```
<!DOCTYPE html>
<html>
        <head>
                <title>My Website</title>
        </head>
        <body>
                <p>Welcome to my Website</p>
```

```
                <form name=”login” action=”process.php” method=”post”>
                Username: <input type="text" name="username"><br />
                Password: <input type="password" name="password">
<br />
                <input type="submit" value="Login">
                </form>
        </body>
</html>
```

Index.html in a browser



As you can see that its not much of a web page. We have not used any fancy layouts, themes or style-sheets as these will be covered in the next Unit dealing with CSS. However, for now all we need to know is that we are building from scratch and that too without using any content management systems. Therefore, its raw code that we will be working with.

2.7 LET US SUM UP

In this unit, we have tried to get acquainted briefly with HTML. Though we have covered only a few of the topics in HTML, these topics are intended to inspire and point you in the direction to further explore. This unit was not intended to provide you with an exhaustive in-depth on the topic but merely to introduce you to some of the basic concepts of HTML.

What we have learned in this unit:

- Hypertext Markup Language (HTML) and its components
- Document Object Model (DOM)
- How to create a simple web page and add a login form.

CHECK YOUR PROGRESS

Q1: HTML markup is used to describe

(a) Web Pages (b) Web Addresses (c) Web Resources (d) Web Links

Q2: Which of the following is an expansion of the abbreviation HTML?

(a) Hyperlink Markup Language (b) Hypertext Makeup Language

(c) Hypertext Markup Language (d) Hyperlink Makeup Language

Q3: Which of the following is a correct syntax for enclosing comments in HTML?

(a) <! this is a html comment --> (b) >!-- this is a html comment --<

(c) <!-- this is a html comment --> (d) <!-- this is a html comment -!>

Q4: Which HTML tags can be used to make the enclosed text bold?

Q5: Which HTML tags can be used to make the enclosed text subscript?

Q6: Which HTML tags define the HTML document body?

Q7: Which of the following HTML elements have empty content?

(a) empty (b) empty

(c) <u>empty</u> (d)

Q8: Which of the following HTTP methods can be used while sending form data?

(a) GET (b) SET (c) POST (d) STOP

Q9: Which of the following HTML list tags produces a bulleted list of items?

(a) Ordered List Tag (b) Unordered List Tag

(c) Description List Tag (d) All the above options (a), (b), (c).

Q10: Which of the following table attribute is used to specify the space between the cell contents and its borders?

(a) cellpadding (b) cellspacing (c) padding (d) spacing

2.8 FURTHER READINGS

- Julie C. Meloni and Michael Morrison, Sams Teach Yourself HTML and CSS in 24 Hours, Pearson Sams, ISBN-13 9780672330971, ISBN-10 0672330970, 2009

- Navneet Mehra and Bunny Mehra, Website Development Using HTML & CSS: A Practical Step-by-Step Guide to Develop e-Commerce Store, Unicorn Books, ISBN-13 9788178063096, ISBN-10 8178063093, 2012

- https://tools.ietf.org/html/rfc2616

- http://www.w3.org/standards/techs/html#w3c_all

2.9 ANSWERS TO CHECK YOUR PROGRESS

1. (a)

2. (c)

3. (c)

4.

5.

6. <body></body>

7. (d)

8. (a), (c)

9. (b)

10. (a)

2.10 MODEL QUESTIONS

1. Which HTML tag represents the root of an HTML document?
2. What is the purpose of the “<noscript></noscript>” HTML tags?
3. What are HTML list tags?
4. Which HTML tags are used to specify a table header?
5. Which HTML image attribute specifies an alternate text for an image, in case the image cannot be displayed?
6. Which are the HTTP methods used while sending form data?
7. Which HTML DOM object is the root node of the HTML document?
8. What is the purpose of the "src" attribute in the <img> tag?
9. What does the <title> tag specify?
10. What is the unit of measure for the table border attribute assignment?

****

UNIT 3: CSS

UNIT STRUCTURE

3.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the basics of CSS
- create your own .css files

- use the custom .css files in web pages.

3.2 INTRODUCTION

Cascading Style Sheets or CSS is a simple styling language for adding style like fonts, colours, spacing or size to Web documents. CSS is the basic technique to separate the appearance of a web page from the content represented in HTML. The styling information is usually stored in external ".css" files. This unit aims to provide a basic understanding of CSS and its components for styling web pages.

3.3 CASCADING STYLE SHEETS (CSS) AND ITS COMPONENTS

CSS styling information is stored in ".css" files which contains a set of rules that matches the HTML elements in a HTML document that is targetted for styling. Each CSS rule consists of one or more **selectors** (i.e., the HTML elements you want to style), separated with commas (,), and a declaration block enclosed in curly braces ({ }). A declaration block contains a list of **property statements** (the style attribute you want to change). Each property has a label and a value, separated with a colon (:). A property statement ends with a semicolon (;).

CSS Syntax:
Example: Single style declaration
selector { property: value; }

Example: Multiple style declarations
selector { property1: value1; property2: value2; }

Though there are other methods (like inline style and embedded style) that can also be used to apply CSS styling apart from using external

style sheets. We will use the external style sheet method in our examples.

3.3.1 Inline Style

Inline style is the style attached to one specific HTML element. The style is specified directly in the start tag as a value of the “style” attribute and will apply exclusively to the specified element.

Example:

index.html

```
<html>
	<head>
		<title>My Title</title>
	</head>
	<body>
		<p style="font-size: large; color: blue;">Inline Style</p>
	</body>
</html>
```

3.3.2 Embedded Style

Embedded style is the style attached to one specific HTML document. The style information is specified as a content of the "style" element inside the "head" element and will apply only to the document it is contained in. Here, the styling rules are written as an HTML comment to hide the content in browsers that do not have CSS support.

Example:

index.html

```
<html>
	<head>
		<style type="text/css">
```

```
		<!--
			p { font-size: large; color: blue; }
		-->
		</style>
	</head>
	<body>
		<p>Embedded Style</p>
	</body>
</html>
```

3.3.3 External Style Sheet

External Style Sheet is a ".css" file containing style information which can be linked with any number of HTML documents. This is a very convenient way of formatting multiple HTML documents as well as restyling web pages by editing just one single ".css" file. The file is linked with HTML documents via the "link" element inside the "head" element. The files containing the style information must have the ".css" extension.

Example:

mystyle.css

```
p { font-size: large; color: blue; }
```

index.html

```
<html>
	<head>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
	</head>
	<body>
		<p>External Style Sheet</p>
	</body>
</html>
```

3.3.4 Imported Style Sheet

Imported style is a style that can be imported to and combined with another style. This allows creating one main style sheet containing declarations that apply to the whole site and sub style sheets containing declarations that apply to specific elements or documents that may require additional styling. By importing sub style sheets to the main style sheet a number of sources can be combined together into one.

To import a style sheet or style sheets include the @import notation or notations in the "style" element. The "@import" notations must come before any other declaration. If more than one sheet is imported they will cascade in the order that they are imported - the last imported style sheet will override the next last; the next last will override the second last, and so on. If the imported style is in conflict with the rules declared in the main sheet then it will be overridden.

Example:

mystyle.css

```
p { font-size: medium; color: blue; }
```

myotherstyle-part1.css

```
h1 { font-size: large; color: red; }
```

myotherstyle-part2.css

```
h2 { font-size: large; color: green; }
```

index.html

```
<html>
	<head>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
		<style type="text/css">
		<!--
```

```
                @import url(myotherstyle-part1.css);
                @import url(myotherstyle-part2.css);
        -->
        </style>
    </head>
    <body>
        <h1>Styling from myotherstyle-part1.css</h1>
        <h2>Styling from myotherstyle-part2.css</h2>
        <p>Styling from mystyle.css</p>
    </body>
</html>
```

3.3.5 Ruleset

The CSS ruleset is one of the CSS rules that identifies a selector or selectors and declares the style which is to be attached to that specific selector or selectors. For example p { font-size: medium; color: blue; } is a CSS ruleset consisting of two parts. In our example, "p" is the selector and "{ font-size: medium; color: blue; }" is the declaration.

Example:
p { font-size: medium; color: blue; } is the **CSS ruleset**.
{ font-size: medium; color: blue; } is the **CSS declaration**.
font-size and **color** are the **CSS properties**.
medium and **blue** are the respective **CSS values** of the CSS properties.

3.3.6 @ rule

The @ rule is one of the CSS rules that applies to the whole style sheet and not just to a specific selector like in the CSS ruleset. @ rules begin with the "@" symbol followed by a keyword made up of letters "a to z", "A to Z", digits "0 to 9", dashes and escaped characters. For example, @import, @font-face, etc.

3.3.7 Class Selector

The CSS class selector is a stand alone class to which a specific style is declared. Using the "class" attribute the declared style can then be associated with any HTML elements. The class selectors are created by a period followed by the class name. The name can contain characters "a to z", "A to Z", digits "0 to 9", period, hyphen, escaped characters, Unicode characters 161-255, as well as any Unicode character as a numeric code, however, they cannot start with a dash or a digit. It is a good practice to name classes according to their function.

Example:

mystyle.css

```
.address {font:60%;}          /* this is a class selector named "address" */
```

index.html

```
<html>
        <head>
                <link rel="stylesheet" href="mystyle.css" type="text/css">
        </head>
        <body>
                <div class="address">This is my Address</div>
                <p class="address">This is my Address too</p>
        </body>
</html>
```

3.3.8 ID Selector

The CSS id selector is an individually identified selector to which a specific style is declared. Using the "id" attribute of an HTML element the declared style can then be associated with that specific HTML element, so as to differentiate it from all the other HTML elements in the HTML document. The CSS id selector is created by using the “#” character followed by the selector's

name. The name can contain characters "a to z", "A to Z", digits "0 to 9", period, hyphen, escaped characters, Unicode characters 161-255, as well as any Unicode character as a numeric code, however, they cannot start with a dash or a digit.

Example:

mystyle.css

```
#myid {color: green; background:black;}/*This is a id selector named "myid"*/
```

index.html

```
<html>
	<head>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
	</head>
	<body>
	<p id="myid">This HTML element is uniquely identified as myid</p>
	</body>
</html>
```

3.3.9 Contextual Selector

The CSS contextual selector is a selector that addresses specific occurrence of an HTML element. It is a string of individual selectors separated by white spaces and a search pattern, where only the last element in the pattern is addressed if it matches the specified context.

Example:

mystlye.css

```
td p code {color:blue;}
```

The element "code" will be displayed in blue only if it occurs in the context of the element "p" which must occur in the context of the element "td".

index.html

```
<html>
	<head>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
	</head>
	<body>
	<table>
	<tr>
		<td>
		<p>
			<code>This text is displayed in blue</code>
		</p>
		</td>
	</tr>
	</table>
	</body>
</html>
```

Example:

mystyle.css

```
td p code, h1 em {color:green;}
```

The element "code" will be displayed in green as described above and the element "em" will also be green, but only if it occurs in the context of the element "h1"

index.html

```
<html>
	<head>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
	</head>
	<body>
	<h1><em>This heading is displayed in green</em></h1>
	<table>
	<tr>
		<td>
```

```
                <p>
                        <code>This text is displayed in green too</code>
                </p>
                </td>
        </tr>
        </table>
        </body>
</html>
```

Example:

mystyle.css

```
p .address {color:blue;}
```

Any element with the class "address" will be displayed in blue but only if it occurs in the context of the element "p"

index.html

```
<html>
        <head>
                <link rel="stylesheet" href="mystyle.css" type="text/css">
        </head>
        <body>
        <table>
        <tr>
                <td>
                <p>
                <code class=”address”>This text is displayed in blue</code>
                </p>
                </td>
        </tr>
        </table>
        </body>
</html>
```

Example:

mystyle.css

```
p .address [lang]{color:green;}
```

Any element with the attribute "lang" will be displayed in green but only if it's class is "address" and it occurs in the context of the element "p"

index.html

```
<html>
	<head>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
	</head>
	<body>
	<table>
	<tr>
		<td>
	<p>
	<code class=”address” lang=”en”> This text is displayed in green</code>
	</p>
		</td>
	</tr>
	</table>
	</body>
</html>
```

3.3.10 Attribute Selector

The CSS attribute selector is a selector defined by the following

(a) the attribute set to element(s)

Example:

mystyle.css

```
a[title] {text-decoration:none;}
```

All the "a" elements containing the "title" attribute will not be underlined.

index.html

```
<html>
	<head>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
	</head>
	<body>
		<a title=”mytitle” href=””>Not Underlined</a>
		<a href=””>Underlined</a>
	</body>
</html>
```

Example:

mystyle.css

```
a[class="myclass"] {text-decoration:none;}
```

All the "a" elements classed as "myclass" will not be underlined.

```
<html>
	<head>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
	</head>
	<body>
		<a class=”myclass” href=””>Not Underlined</a>
		<a href=””>Underlined</a>
	</body>
</html>
```

(b) the attribute and value(s)

Example:

mystyle.css

```
a[title="myvalue"] {text-decoration:none;}
```

All the "a" elements containing the "title" attribute with a value that is an exact match of the specified "myvalue", will not be underlined.

index.html

```
<html>
```

```
        <head>
                <link rel="stylesheet" href="mystyle.css" type="text/css">
        </head>
        <body>
                <a title=”myvalue” href=””>Not Underlined</a>
                <a title=”myothervalue” href=””>Underlined</a>
        </body>
</html>
```

(c) the attribute and value parts

Example:

mystyle.css

```
a[title~="myvalue"] {text-decoration:none;}
```

All the "a" elements containing the "title" attribute with a value containing the specified word "myvalue", will not be underlined.

index.html

```
<html>
        <head>
                <link rel="stylesheet" href="mystyle.css" type="text/css">
        </head>
        <body>
                <a title=”myvalue” href=””>Not Underlined</a>
                <a title=”myvalue other” href=””>Not Underlined</a>
                <a title=”myother value” href=””>Underlined</a>
        </body>
</html>
```

This section of the unit only covered some of the basic concepts of CSS. In the next section, we will briefly touch upon some of the CSS Properties for styling HTML elements. We will use this later, in the “Applying what we have learnt” section.

3.4 CSS PROPERTIES

Discussing about all the CSS properties is beyond the scope of this unit. Therefore, we will touch upon only some of the ones commonly used to get you started.

3.4.1 Background Properties

background-color

This property specifies the background colour of an HTML element. The colour can be specified as a HEX value "#ff0000", a RGB value "rgb(255,0,0)" or a color name "red".

Example:

```
body {background-color:#eeeeee;}
```

background-image

This property specifies an image as the background of an HTML element.

Example:

```
body {background-image:url("mybackground.png");}
```

background-repeat

The background-image property repeats an image both horizontally and vertically if the "background-repeat" property is not specified. In situations when you need to control the image repeat, the background-repeat property can be used to either repeat the image horizontally using "repeat-x" or vertically using "repeat-y". You could also use "no-repeat" to show the image only once without repeats.

Example:

```
body {background-image:url("mybackground.png"); background-repeat:repeat-x;}
```

background-attachment

This property sets whether a background image is fixed or scrolls with the rest of the page. Commonly used values are scroll and fixed.

Example:

```
body {background-image:url("mybackground.png"); background-repeat:no-repeat; background-attachment:fixed;}
```

background-position

The property specifies the position of the image. The first value is the horizontal position (x) and the second value is the vertical position (y).

Example:

```
body {background-image:url("mybackground.png"); background-repeat:no-repeat; background-position:right top;}
```

3.4.2 Text Properties

color

This property is used to set the colour of the text. The colour can be specified as a HEX value "#ff0000", a RGB value "rgb(255,0,0)" or a color name "red".

Example:

```
body {color:blue;}
```

text-align

This property is used to set the horizontal alignment of a text. Text can be centered, aligned left, aligned right or justified.

Example:

```
p {text-align:justify;}
```

text-decoration

This property is used to set or remove decorations from text. The values could be none, underline, overline, line-through.

Example:

```
a {text-decoration:none;}
```

text-transform

This property is used to specify uppercase and lowercase letters in a text. The values could be none, capitalize, uppercase, lowercase.

Example:

```
p {text-transform:capitalize;}
```

text-indent

This property is used to specify the indentation of the first line of a text.

Example:

```
p {text-indent:20px;}
```

3.4.3 Border Properties

border-style

This CSS property specifies what kind of border to display for the HTML element. The values can be none, dotted, dashed, solid, double, groove, ridge, inset, outset.

Syntax for specifying a single type of border for all sides:

border-style:border-value

Syntax for specifying different borders for each side:

border-style:top-border-value right-border-value bottom-border-value left-border-value

Example:

```
p .none {border-style:none;}

p .dotted {border-style:dotted;}
```

```
p .multiple {border-style:dotted double solid dashed;}
```

border-width

This CSS property is used to set the width of the border that was set using the "border-style" property. The values can be either in pixels or thin, medium, thick.

Example:

```
p .dotted {border-style:dotted; border-width:2px;}
```

border-color

This CSS property is used to set the color of the border that was set using the "border-style" property. The colour can be specified as a HEX value "#ff0000", a RGB value "rgb(255,0,0)" or a color name "green".

Example:

```
p .dotted {border-style:dotted; border-width:2px; border-color:#00ff00}
```

There are a whole lot of CSS properties that can be used and this unit is in no way intended to cover all of them. You are encouraged to use the online resources available at your disposal to further explore the CSS properties.

In the next sections, we will apply what we have covered in the previous sections of this unit, to create a CSS file and then to use it to style a web page.

3.5 APPLYING WHAT WE HAVE LEARNT III - CREATING CSS FILE

To start creating a CSS file, all we really need is a text editor. Though you could use other advanced editors, it is recommended that you use a simple text editor to get your concepts clear.

Let us start, by creating a CSS document named “mystyle.css” with the contents as show below. Notice that the file extension we are using is “.css”.

mystyle.css

```
/* login-form class definition */
.login-form {
        color:darkred; /* darkred text colour */
        text-transform:uppercase; /* text in uppercase */
        border-style:dotted; /* dotted border */
        border-width:2px; /* 2 pixels border width */
        border-color:#ff0000;  /* red border colour */
        background-color:#cccccc; /* background colour is a shade of grey */
}

/* apply to all the td elements of the login-form class */
.login-form td {
   text-align:center; /* center align td elements */
   vertical-align:middle;  /* vertically middle align td elements */
}

/* apply to all the input elements of the login-form class */
.login-form input {
        color:darkred; /* input elements have darkred text colour */
        background-color:#eeeeee; /* input elements have grey background */
        height:25px; /* input elements have 25 pixels height */
        width:200px; /* input elements have 200 pixels width */
```

```
}
```

We have created a CSS file which will be used in the next section to add some styling to the login form we had created previously in Unit 2, “Applying What We Have Learnt - II”.

3.6 APPLYING WHAT WE HAVE LEARNT IV – USING CSS IN A WEB PAGE

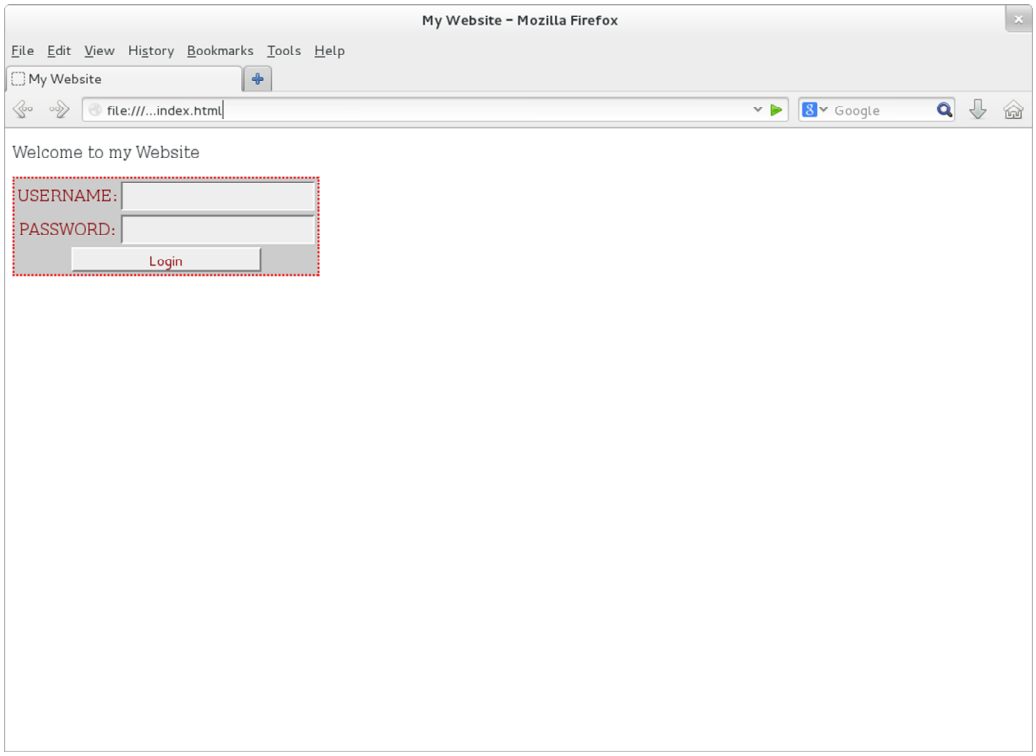
Now, we add the “mystyle.css” file we had created in the previous section to the “index.html” file that we had created in the previous Unit 2, “Applying What We Have Learnt – II” section, as shown below.

index.html

```
<!DOCTYPE html>
<html>
	<head>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
		<title>My Website</title>
	</head>
	<body>
		<p>Welcome to my Website</p>
<form action="process.php" method="post">
<table class="login-form">
<tr>
	<td>Username:</td><td><input type="text" name="username"></td>
</tr>
<tr>
	<td>Password:</td><td><input type="password" name="password"></td>
</tr>
<tr>
	<td colspan="2"><input type="submit" value="Login"></td>
```

```
</tr>
</table>
</form>
        </body>
</html>
```

Index.html in a browser



As you can see that the styling that was defined in the “mystyle.css” file has been successfully applied to the “index.html” page. We can go on to further improve the aesthetics of this page. However, the purpose of this unit was to expose you to the basics of CSS.

3.7 LET US SUM UP

In this unit, we have briefly discussed the basics of CSS. Though we have covered only a few of the topics in CSS, these topics are intended to inspire and point you in the direction to further explore. This unit was not intended to provide

you with an exhaustive in-depth on the topic but merely to introduce you to some of the basic concepts of CSS.

What we have learned in this unit.

- CSS basics
- Creating a CSS file
- Using CSS in a Web page

CHECK YOUR PROGRESS

Q1: CSS is used for styling

(a) Web Pages (b) Web Addresses (c) Web Resources (d) Web Links

Q2: Which of the following is an expansion of the abbreviation CSS?

(a) Cascading Style Sheets (b) Cascading Style System

(c) Cascading System Style (d) Cascading Sheets Style

Q3: Which of the following is a correct syntax for enclosing comments in CSS?

(a) <! this is a CSS comment --> (b) /* this is a CSS comment */

(c) /* this is a CSS comment --> (d) <!-- this is a CSS comment */

Q4: What does a CSS rule consists of?

Q5: What is a CSS selector in a CSS rule?

Q6: What does the CSS declaration block in a CSS rule contain?

Q7: How is a CSS property statement written?

Q8: Which of the following is true for the CSS Inline Style?

(a) Inline style is the style that affects only one specific HTML element.

(b) Inline style is the style that affects all HTML element(s) in the document.

(c) Inline style is the style that affects all the documents of a website.

(d) none of the above.

Q9: Which of the following statements uses CSS External Style Sheets?

(a) <style type="text/css"><!-- p { font-size: large; color: blue; } --></style>

(b) Inline Style

(c) <link rel="stylesheet" href="mystyle.css" type="text/css">

(d) All the above options (a), (b), (c).

Q10: Which of the following will be affected by the CSS ruleset

a[title~="myvalue"] {text-decoration:none;}

(a) Not Underlined

(b) Not Underlined

(c) Underlined

(d) None of the above.

3.8 FURTHER READINGS

- Julie C. Meloni and Michael Morrison, Sams Teach Yourself HTML and CSS in 24 Hours, Pearson Sams, ISBN-13 9780672330971, ISBN-10 0672330970, 2009

- Navneet Mehra and Bunny Mehra, Website Development Using HTML & CSS: A Practical Step-by-Step Guide to Develop e-Commerce Store, Unicorn Books, ISBN-13 9788178063096, ISBN-10 8178063093, 2012

- http://www.w3.org/Style/CSS/Overview.en.html

- http://www.w3.org/Style/Examples/011/firstcss

3.9 ANSWERS TO CHECK YOUR PROGRESS

1. (a)

2. (a)

3. (b)

4. A CSS rule consists of a selector and a declaration block. In case of multiple selectors, the selectors are separated with commas. Example: p {color:#ff0000;}

5. CSS selectors are patterns used to select the HTML element(s) you want to style in HTML documents. Example: p

6. The CSS declaration block contains a list of CSS property statements. It is enclosed with curly braces. Example: {color:#ff0000;}

7. A CSS property statement has a label and a value, separated with a colon and ends with a semicolon. Example: color:#ff0000;

8. (a)

9.(c)

10. (a), (b)

3.10 MODEL QUESTIONS

1. What is CSS and what is it used for?
2. Which file extension is used to store CSS rules?
3. What is a CSS ruleset?
4. What is the CSS @ rule used for?
5. What is a CSS Contextual Selector?
6. What does the CSS Attribute Selector do?
7. Which CSS Style could be used to import from multiple style sheets, to combine into a single style?
8. What is the CSS color property used for?

9. What is the CSS background-color property used for?
10. What is the CSS background-attachment property used for?

*****

UNIT 4: JAVASCRIPT

UNIT STRUCTURE

4.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the basics of JavaScript
- write simple JavaScript code
- use JavaScript in web pages

4.2 INTRODUCTION

JavaScript is a scripting language that runs on the web browser being used to view web pages containing JavaScripts. Almost all web pages that are developed today, contain JavaScript in them for enhancing the functionality or user experience. In certain cases, if JavaScript is disabled in the browser, the web page content functionality maybe limited or unavailable. JavaScript code is usually written either within the HTML pages or stored in external ".js" files. This unit aims to provide a basic understanding of JavaScript and its components for scripting web pages.

We will be using Mozilla Firefox (version 27+) as our browser to test all our HTML and JavaScript codes.

4.3 JAVASCRIPT

JavaScript is written enclosed within the HTML "<script></script>" tags which should be used within the "<body></body>" or the "<head></head>" sections of the HTML page containing the JavaScripts.

Let us go ahead and create an "index.html" file and include the code as shown below.

Index.html

```
<!DOCTYPE html>
```

```
<html>
	<head>
	<meta charset="UTF-8">
	</head>
	<body>
	  <script>
	  var x = 7;
	  var y = 11;
	  var z = x + y;
	  document.write("The sum (x + y), when x = ",x," and y = ",y," is ",z);
	  </script>
	</body>
</html>
```

You can also include external ".js" files using the "src" property of the "<script>" tag. We will use the include external ".js" file method in all the unit examples. That way, we know where our JavaScripts are and do not need to search within all the HTML pages.

Let us create a "myjavascripts.js" file with the code show below.

myjavascripts.js

```
var x = 7;  //variable x declaration and value assignment
var y = 11; //variable y declaration and value assignment
var z = x + y; //variable z declaration and value assignment
```

And include it in the "index.html" file as shown below.

Index.html

```
<!DOCTYPE html>
<html>
	<head>
	  <meta charset="UTF-8">
	  <!-- including an external JavaScript file -->
	  <script src="myjavascripts.js"></script>
	</head>
```

```
    <body>
        <script>
        document.write("The sum (x + y), when x = ",x," and y = ",y," is ",z);
        </script>
    </body>
</html>
```

In the following sub sections we will take a look briefly at some of the components of the JavaScript language.

4.3.1 Data Types

strings

The string data type are sequences of characters, like the letters a to z, spaces, and even numbers.

Examples:

"KKHSOU", "2020", "What is your name?"

numbers

The number data type are quantities, like the numbers 0 to 9 on which you can perform math operations.

Examples:

20.20, 0.14, 1024

booleans

The boolean data type is either true or false, like comparing two numbers returns either a true or false result.

Examples:

1 > 2 returns false, 3 < 4 returns true

4.3.2 Comparison and Logical Operators

> : Greater than [*Example:* 2 > 1]
< : Less than [*Example:* 1 < 2]
<= : Less than or equal to [*Example:* x <= 10]
>= : Greater than or equal to [*Example:* x >= 5]
== : Equal to [*Example:* y == 0]
!= : Not equal to [*Example:* y != 1]
&& : Logical AND [*Example:* x == 1 && y != 1]
|| : Logical OR [*Example:* x >= 1 || x <= 5]
! : Logical NOT [*Example:* !(x == y)]

4.3.3 Mathematical Operators

( ) : controls the order of operations. The values within the parenthesis gets evaluated first. [*Example:* 5 + (3 * 2)]
***** : multiplication operator [*Example:* 3 * 2]
/ : division operator [*Example:* 4 / 2]
- : subtraction operator [*Example:* 3 - 2]
+ : addition operator [Example: 5 + 6]
% : modulo operator [Example: 4 % 2]

4.3.4 Comments

It is always a good idea to comment your code to make it readable and understandable by programmers reading your code. The JavaScript comments are written using the "//" characters.

Example:

```
//this is a comment in JavaScript
```

4.3.5 The document.write( ) method

The "document.write( )" function is used to write the output in HTML, everything that is contained within the parenthesis “( )”.

Example:

```
document.write("Hi there!"); //prints “Hi there!” in the web page
```

4.3.6 The console.log( ) method

The "console.log( )" function prints into the web console whatever we put in the parentheses.

Example:

```
console.log("Hi there!"); //prints "Hi there!" in the web console.
console.log(1 + 1); //prints "2" in the web console.
```

The Mozilla Firefox web console can be accessed via the short-cut keys “**shift + ctrl + k**” or from the Mozilla Firefox Menu “**Tools → Web Developer → Web Console**”.

4.3.7 Variables

A variable in JavaScript stores the value of the variable, whether that is a number or a string.

Syntax:

```
var variableName = variableData;
```

Example:

```
var myName = "KKHSOU";
var myAge = 25;
console.log(myName.length); //prints "6"  in the web console.
console.log(myAge); //prints 25 in the web console.
```

4.3.8 Length

Returns the length of a string.

Example:

```
document.write("KKHSOU".length); //prints out "6"
```

4.3.9 Substring

The substring is used to extract part of a string.

Syntax:

"string".substring(x, y) where, x is where you start and y is where you finish extracting from the string.

Example:

K	K	H	S	O	U
0	1	2	3	4	5

```
document.write("KKHSOU".substring(2, 4)); //prints out "HS"
```

4.3.10 Conditional Statements - if

Conditional statements can be used when you want to execute block of codes based on specific conditions.

if statement

An if statement is made up of the "if" keyword, a condition specified within the "( )" brackets and followed by code within a pair of curly braces "{ }". If the answer to the condition is true, the code inside the curly braces will run.

Syntax:

if (condition)

{

statement A //execute if condition is true

}

Example:

```
if ( myAge < 13 )
{
        document.write( "Infant" );
}
```

if and **else statement**

On occations when both the true and false conditions need to be handled, the "else" keyword can be used alongwith the "if" keyword.

Syntax:

if (condition)

{

statement A //execute if condition is true

}

else

{

statement B //execute if condition is false

}

Example:

```
if ( myAge < 13 )
{
        document.write( "Infant" );
}
else
{
        document.write( "Not Infant" );
}
```

if, else if and **else statement**

On occations when multiple conditions need to be handled, the "else if" keyword can be used alongwith the "if" and “else” keyword.

Syntax:

```
if ( condition A )
{
        statement A //execute if condition A is true
}
else if ( condition B )
{
        statement B //execute if condition B is true
}
else
{
        statement C //execute if both conditions A and B are false
}
```

Example:

```
if ( myAge < 13 )
{
        document.write( "Infant" );
}
else if ( myAge > 12  && myAge < 19 )
{
        document.write( "Teenager" );
}
else
{
        document.write( "Adult" );
}
```

4.3.11 Loops - for

The **for** loop can be used for execute a block of code multiple times. In the **for** loop syntax, the *statement A* is executed before the *code block* starts, the

statement B defines the condition for running the *code block* and the *statement C* is executed each time after the *code block* has been executed.

Syntax:

for (*statement A***;** *statement B***;** *statement C* **)**

{

code block

}

Example:

```
for ( var i=1; i<=10; i++ )
{
        document.write( "The value of i is " + i + "<br />" );
}
```

4.3.12 Functions

A **function** is a block of code that can be executed when an event occurs by JavaScript code. JavaScript is case sensitive therefore the function keyword must be written in lowercase letters, and the function name must be called with the exact same function name. While calling a function, you can also pass along some values called arguments or parameters to the function. These arguments will then be used inside the function and in case of multiple arguments you can separate them with commas.

Syntax:

function *function-name***(***arguments***)**

{

code block

}

Example:

```
//Function myAddFunction
function myAddFunction( x, y )

{
```

```
        var z = x + y;
        document.write( "Sum of x = "+x+" and y = "+y+" is "+z );
}

//Call to myAddFunction
myAddFunction( 2, 3 );
```

4.4 HTML DOM AND JAVASCRIPT

In the HTML Document Object Model, the "document" object represents the HTML page and is the root of all other objects in the HTML page. To access objects in an HTML page we have to start at the "document" object.

We will briefly discuss some of the JavaScript methods and properties of the HTML DOM and use them in our scripts.

4.4.1 Finding HTML Elements

To access any HTML element using JavaScript, we can utilize the "id" attribute of that specific element. The JavaScript **getElementById** method can then be used in this case to access any element in the HTML document. To get or set the contents of any HTML element we can utilize the **innerHTML** property of the getElementById method.

Example:

```
document.getElementById("myid").innerHTML;
```

4.4.2 Changing HTML Elements

JavaScript can be used to change the content of HTML elements. The JavaScript **document.write()** method can be used to write directly to the HTML output stream. Therefore, if used after the HTML document is loaded it will overwrite the document.

Example:

```
document.write("KKHSOU".length);
```

To modify the content of an HTML element, we can use the **innerHTML** property as shown below.

Example:

```
document.getElementById("myid").innerHTML="My Modified Content";
```

We can also modify the attribute of an HTML element using the **attribute** property as shown below.

Example:

```
document.getElementById("myid").href="http://mywebsite/";
```

4.4.3 DOM events

The HTML Document Object Model allows the execution of code when an event occurs. These events are generated by the browser when the HTML elements are interacted upon by actions such as when an element is clicked on, the selected field in a drop down menu has changed, the mouse pointer hovers over an element, the HTML page is loaded, etc.

onclick

A user clicks on an HTML element to execute a JavaScript code.

Usage:

onclick="JavaScript Code"

Example:

```
<p onclick="this.innerHTML='You clicked me'">Click me!</p>
```

onchange

A user changes the value of an HTML element to execute a JavaScript code.

Usage:

onchange="JavaScript Code"

Example:

```
<select id="myid" onchange="this.style.background='red'">
        <option value="1">1</option>
```

```
        <option value="2">2</option>
        <option value="3">3</option>
</select>
```

onmouseover, onmouseout

A user navigates the mouse pointer over or out of an HTML element to execute a JavaScript code.

Usage:

onmouseover="JavaScript Code"

onmouseout="JavaScript Code"

Example:

```
<p onmouseover="this.innerHTML='You are over me'"
onmouseout="this.innerHTML='Hi Again'">Hi</p>
```

We have briefly discussed a few of the events with examples in this section to get you started. However, there are a lot more events that can be used to trigger JavaScript code execution and we will discuss about these as and when we come across them in the later units.

4.5 APPLYING WHAT WE HAVE LEARNT V – CREATING A JAVASCRIPT FILE

To start creating a JavaScript ".js" file, all we really need is a text editor. Though you could use other advanced editors, it is recommended that you use a simple text editor to get your concepts clear.

Let us start, by creating a .js file named "myjavascripts.js" with the contents as show below. Notice that the file extension we are using is ".js". Here we are utilizing the JavaScript *match()* method which searches a string for a match against a regular expression to read the characters typed into the Username input box and display whether the values are Alphabets or Numbers.

myjavascripts.js

```
//function checkUsername checks the username input; whether blank,
//numbers, letters or special characters.
function checkUsername()
{
	var username;
	username = document.getElementById("username").value;
	if ( username=="" )
	{
	document.getElementById("messageBox").value = "Blank Username";
	}
	else if ( username.match(/[a-z]/i) )
	{
	document.getElementById("messageBox").value = "Alphabets";
	}
	else if ( username.match(/[0-9]/) )
	{
	document.getElementById("messageBox").value = "Numbers";
	}
	else
	{
	document.getElementById("messageBox").value = "Special Characters";
	}
}
```

We have created a JavaScript file which will be used in the next section to add some dynamism to the login form we had created previously in Unit 3, “Applying What We Have Learnt - IV”.

3.6 APPLYING WHAT WE HAVE LEARNT VI – USING JAVASCRIPT IN A WEB PAGE

Now, we add the “myjavascripts.js” file we had created in the previous

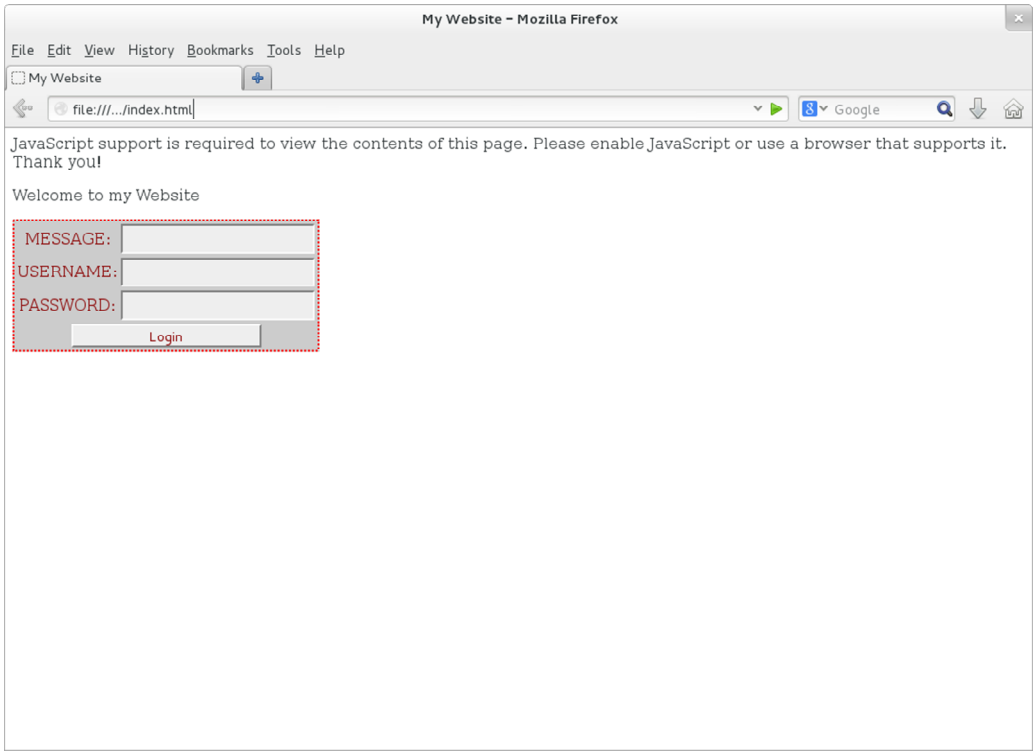
section to the “index.html” file that we had modified in the previous Unit 3, “Applying What We Have Learnt – IV” section, as shown below.

index.html

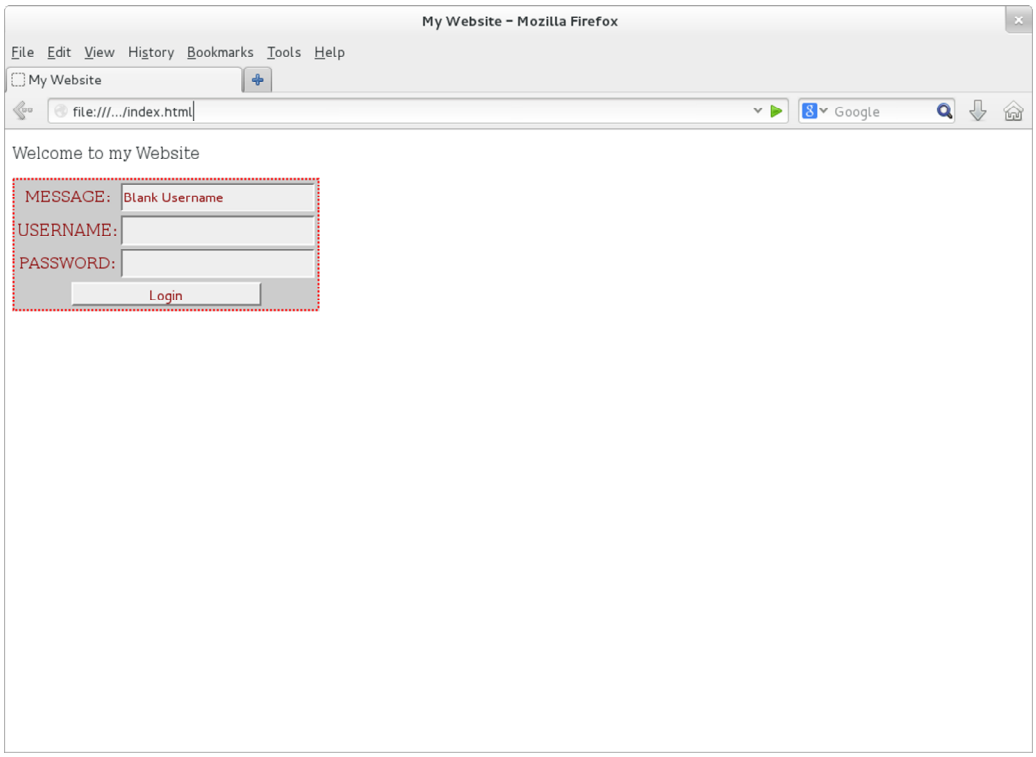
```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8">
		<noscript>JavaScript support is required to view the contents of this page. Please enable JavaScript or use a browser that supports it. Thank you!</noscript>
		<script src="myjavascripts.js"></script>
		<link rel="stylesheet" href="mystyle.css" type="text/css">
		<title>My Website</title>
	</head>
	<body>
		<p>Welcome to my Website</p>
		<form action="process.php" method="post">
		<table class="login-form">
		<tr>
		<td>Message:</td><td><input id="messageBox" type="text" name="messageBox" readonly></td>
		</tr>
		<tr>
		<td>Username:</td><td><input id="username" type="text" name="username" onkeyup="checkUsername()"></td>
		</tr>
		<tr>
		<td>Password:</td><td><input id="password" type="password" name="password"></td>
		</tr>
		<tr>
		<td colspan="2"><input id="login" type="submit" value="Login"></td>
```

```
                </tr>
                </table>
                </form>
        </body>
</html>
```

Index.html in a browser that has JavaScript disabled or does not support it



Index.html in a browser that has JavaScript enabled



As you can see that now the Message input box content changes depending on what you type into the Username input box and provides some dynamism to our “index.html” page. This example should intrigue you to further explore the possibilities of dynamism that can be applied to a web page. We have not covered all the topics extensively however, the purpose of this unit was to expose you to the basics of JavaScript and use it on HTML pages.

4.7 LET US SUM UP

In this unit, we have briefly discussed the basics of JavaScript. Though we have covered only a few of the topics in JavaScript, these topics are intended to inspire and point you in a direction to further explore. This unit was not intended to provide you with an exhaustive in-depth on the topic but merely to introduce you to some of the basic concepts of JavaScript and use them in web pages.

What we have learned in this unit.

- JavaScript basics
- Creating a JavaScript file
- Using JavaScript in a Web page

CHECK YOUR PROGRESS

Q1: JavaScript is

(a) a web page (b) a scripting language

(c) a css file (d) a web page with scripts

Q2: Which of the following file extension(s) is(are) used for a JavaScript file?

(a) .html (b) .css

(c) .java (d) .js

Q3: Which of the following is a correct syntax for enclosing comments in JavaScript?

(a) <! this is a JavaScript comment --> (b) /* this is a JavaScript comment */

(c) // this is a JavaScript comment (d) this is a JavaScript comment //

Q4: When writing JavaScripts within an HTML document which pair of HTML tags have to be used?

(a) <code></code> (b)<javascript></javascript>

(c) <noscript></noscript> (d) <script></script>

Q5: Which pair of HTML tags are used to display messages on the web page for browsers that either have JavaScript disabled or does not support it?

(a) <code></code> (b)<javascript></javascript>

(c) <noscript></noscript> (d) <script></script>

Q6: Which JavaScript method can be used to search for HTML DOM elements based on the element “id” attribute?

Q7: When a user navigates the mouse pointer over an HTML element, which of the following HTML DOM events are triggered?

(a) onmouseout (b) onkeyup

(c) onmouseover (d) onclick

Q8: Which of the following JavaScript modifies the content of the HTML element with the attribute id="myid"?

(a) document.getElementByTag("id").innerHTML="Modified Content";

(b) document.getElementById("id").innerHTML="Modified Content";

(c) document.getElementById("myid").innerHTML="Modified Content";

(d) document.getElementByTag("myid").innerHTML="Modified Content";

Q9: Which of the following DOM event is triggered when a user changes the value of an HTML element?

(a) onchange (b) onclick

(c) onkeyup (d) onfocus

Q10: Which of the following gets executed when the **if** condition is true?

(a) The code block within the curly braces "{ }" of the **if** statement

(b) The code block outside the curly braces "{ }" of the **if** statement

(c) The code block inside the braces "( )" of the **if** statement

(d) None of the above.

4.8 FURTHER READINGS

- Michael Moncur, "Sams Teach Yourself JavaScript in 24 Hours 4th Edition", Dorling Kindersley (RS), ISBN-13 9788131704554, ISBN-10 8131704556, 2006
- David Flanagan, "JavaScript: The Definitive Guide 6th Edition", O'Reilly, ISBN-13 9789350233948, ISBN-10 9350233948, 2011
- https://developer.mozilla.org/en/docs/Web/JavaScript

- http://www.javascriptkit.com/javatutors/index.shtml

4.9 ANSWERS TO CHECK YOUR PROGRESS

1. (b)
2. (d)
3. (c)
4. (d)
5. (c)
6. getElementById( )
7. (c)
8. (c)
9. (a)
10. (a)

4.10 MODEL QUESTIONS

1. What is JavaScript and what is it used for?
2. Which file extension is used to store JavaScript code?
3. Which HTML tags can be used to display text if the web browser does not support JavaScript?
4. What is the JavaScript logical operator "!" used for?
5. In JavaScript, what is the difference between the "=" and the "==" operators?
6. What does the JavaScript **document.write( )** method do?
7. How can you include an external JavaScript file "myjavascripts.js" into an HTML document?
8. If you write JavaScripts inside a HTML document, which HTML tags should you use?
9. What is the syntax for comments in JavaScript?
10. What is a DOM event?

****

UNIT 5: XML AND AJAX

UNIT STRUCTURE

5.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the basics of XML and Ajax
- write simple XML and Ajax code
- use XML and Ajax in web pages

5.2 INTRODUCTION

XML is a markup language that is designed to store and transport data unlike HTML which is designed for displaying data. Ajax allows web pages to be updated asynchronously making it possible to update parts of a web page, without having to reload the entire web page. Without Ajax the entire web page would need to be reloaded for displaying any change in the content. This unit aims to provide the basic understanding of XML and Ajax to build web pages.

We will be using Mozilla Firefox (version 27+) as our browser to test all our HTML, XML and Ajax codes.

5.3 XML

XML stands for the e**X**tensible **M**arkup **L**anguage which is designed to transport and store data just as HTML is designed to display data. Though XML is a markup language like HTML, the tags used in XML are not predefined like HTML and therefore must be defined by the author of the XML document. XML allows the author to define custom tags and custom document structure. However, all the XML elements must have an opening and a matching closing tag and also care should be taken as XML is case sensitive.

The XML document is stored in plain text format with the ".xml" file extension and provides a platform independent way of storing data. This feature makes XML a popular markup language.

Let us go ahead and create an XML document "myxmlfile.xml" and populate it with the contents shown below. We will use this XML file in our examples.

myxmlfile.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
        <book>
                <title>Title A</title>
                <author>Author A</author>
```

```
        <publisher>Publisher A</publisher>
        <year>2001</year>
    </book>
    <book>
        <title>Title B</title>
        <author>Author B</author>
        <publisher>Publisher B</publisher>
        <year>2002</year>
    </book>
    <book>
        <title>Title C</title>
        <author>Author C</author>
        <publisher>Publisher C</publisher>
        <year>2003</year>
    </book>
</library>
```

5.3.1 Declaration

The first line in an XML file should contain the XML declaration, which defines the XML version and the character encoding to be used by an XML parser while parsing the XML document. Modern web browsers have an XML parser in-built and can therefore parse XML documents by default.

In our example of an XML file shown above, the first line of the XML document contains the line **<?xml version="1.0" encoding="UTF-8"?>** which is the **XML declaration**. Notice that the "<?xml ... ?>" pair of tags are used for the declaration.

5.3.2 Root Element

An XML document will contain one single root element that should identify the elements that it contains. Since, XML does not have predefined tags, the

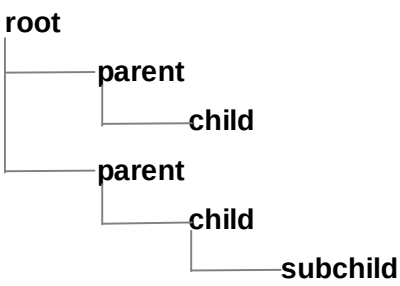
author of an XML document is free to invent the tags to use. However, one should keep in mind that the tag names should be kept consistent throughout the XML document.

In our example above, we have indented our XML code for making the code readable. Therefore, you would notice right away that the pair of tags "**<library> ... </library>**" contains sub elements within it and is the **root element** for this XML document. The XML root element is the parent of all the sub elements in an XML document. All the elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
        <parent>
                <child> ... </child>
        </parent>
        <parent>
                <child>
                        <subchild> ... </subchild>
                </child>
        </parent>
</root>
```

XML document tree



5.3.3 Child Elements

An XML root element will contain sub elements which are called child elements of the root element. Child elements can contain sub elements and so on.

In our example above, all the three sub elements "**<book> ... </book>**" of the root element are the **child elements** of the root element. They are also siblings of each other and each of them contain the child elements "<title> ... </title>", "<author> ... </author>", "<publisher> ... </publisher>", "<year> ... </year>" as well. All the XML elements can have text content and attributes like HTML elements.

5.3.4 Element Attributes

The XML elements can have attributes with name="value" pairs similar to the HTML elements. However, the attribute values must always be quoted.

Example:

```
<book onshelf="no">
    <title>Title C</title>
    <author>Author C</author>
    <publisher>Publisher C</publisher>
    <year>2003</year>
</book>
```

We will not be using XML element attributes in our examples though.

5.3.5 Entity References

Some characters have special meaning in XML and therefore should be replaced with the entity references listed below, if you plan to use them in your XML document.

<	:	<	less than
>	:	>	greater than

& : & ampersand

' : ' apostrophe

": " quotation mark

5.3.6 Comments

The syntax for writing comments in XML is similar to that in HTML. You should always include comments in your codes for making it readable to other developers.

Example:

```
<!-- This is an XML comment -->
```

5.4 AJAX

Ajax stands for **A**synchronous **J**avaScript **A**nd **X**ML. It is not a programming language, but a technique to use existing standards for exchanging data with a server asynchronously and updating portions of a web page without having to reload the entire web page.

Ajax is based on Internet standards, and uses a combination of the *XMLHttpRequest* object to exchange data asynchronously with a web server, *JavaScript* to display or interact with the data, *CSS* to style the data and *XML* format for transferring the data. However, the data formats for transferring data can also be in plain text and HTML. Ajax applications are browser and platform independent.

5.4.1 XMLHttpRequest Object

The XMLHttpRequest is an Application Programming Interface available to web browser scripting languages such as JavaScript and is used to send HTTP or HTTPS requests to a web server and load the server response data back into the script. The XML Http Request object is subject to the browser's

same-origin policy for security reasons. Therefore, XMLHttpRequest requests will only succeed if they are made to the same server that served the original web page.

We need to create an instance of the XMLHttpRequest object before being able to use it. The following example demonstrates how to create an instance of the XMLHttpRequest object with JavaScript.

Example:

```
var myXMLHttp; //This is my variable to store the XMLHttpRequest object
myXMLHttp = new XMLHttpRequest();
```

Though most of the modern browsers have a built in XMLHttpRequest object, it is a good practice to check if the browser supports the XMLHttpRequest object. The following JavaScript example demonstrates how to check if the browser supports the XMLHttpRequest object, and then either create or inform if not supported by the browser.

Example:

```
var myXMLHttp;
if (window.XMLHttpRequest)
{
        myXMLHttp = new XMLHttpRequest();
}
else
{
        throw new Error("Your Browser does not support the XMLHttpRequest object. Please upgrade your browser.");
}
```

5.4.2 Sending Ajax Requests

Once the XMLHttpRequest object has been created, we can use its

open() and **send()** methods to send requests to the web server as shown in the JavaScript example below.

Syntax:

open(**method, url, asynchronous, username, password**);

Here, the first parameter "**method**" is a text string indicating the HTTP request method to use *(GET or POST methods are used generally)*. The second parameter "**url**" is a text string indicating the URL of the HTTP request *(the url has to be pointing to the source of the current document)*. The third parameter "**asynchronous**" is a boolean value indicating whether or not the request will be asynchronous (this is true by default). The fourth and fifth parameters "**username**" and "**password**" respectively may be provided for authentication and authorization if required by the server for the request.

Syntax:

send(**data**);

If the HTTP method specified in the *open()* method is GET, then the parameter "**data**" can be omitted as no data is to be sent, as shown in the example below.

Example:

```
myXMLHttp.open("GET","myajaxtext.txt",true);
//myajaxtext.txt is a file on the web server
myXMLHttp.send();
```

However, if the HTTP method specified in the *open()* method is POST, which is used to submit HTML form data, then the XMLHttpRequest object's **setRequestHeader()** method is also needed before sending the request to the web server. The usage of the setRequestHeader() method is as shown below.

Syntax:

setRequestHeader(**headername**, **value**);

Here, "**headername**" is a text string and specifies the HTTP header name and "**value**" is a text string and specifies the HTTP header value. This method must be invoked for each header that needs to be sent with the request. Any headers attached here will be removed the next time the *open()* method is

invoked in a W3C conforming user agent.

Example:

```
myXMLHttp.open("POST","myajaxapp.php",true);
myXMLHttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
myXMLHttp.send("username=kkhsou&password=abcdefg");
```

5.4.3 Handling Ajax Responses

If the *open()* method of the XMLHttpRequest object is invoked with the third parameter set to "true" for an asynchronous request, then the "**onreadystatechange**" event listener will be automatically invoked for each of the following actions that change the "**readyState**" property of the XMLHttpRequest object, as explained in the paragraph below.

After the *open()* method is invoked successfully, the *readyState* property of the *XMLHttpRequest* object is assigned a value of “1”. After the *send()* method is invoked and the HTTP response headers have been received, the *readyState* property of the *XMLHttpRequest* object is assigned a value of “2”. Once the HTTP response content begins to load, the *readyState* property of the *XMLHttpRequest* object is assigned a value of “3”. Once the HTTP response content has finished loading, the *readyState* property of the *XMLHttpRequest* object is assigned a value of “4”.

The *onreadystatechange* event listener will only respond to state changes which occur after it is defined. To detect states 1 and 2, the *onreadystatechange* event listener must be defined before the *open()* method is invoked. The *open()* method must be invoked before the *send()* method is invoked.

Example:

```
myXMLHttp.onreadystatechange = function() {
if (myXMLHttp.readyState === 4){
```

```
document.getElementById(myId).innerHTML = myXMLHttp.responseText;
//This will write the responseText value to the HTML Element with the attribute id=”myId”
        }
};
```

Combining all the above example codes, the complete functional Ajax code is as shown below.

```
if (window.XMLHttpRequest)
{
        var myXMLHttp = new XMLHttpRequest();
}
else
{
        throw new Error("Your Browser does not support the XMLHttpRequest object. Please upgrade your browser.");
}

myXMLHttp.onreadystatechange = function() {
        if (myXMLHttp.readyState === 4){
        document.getElementById(myId).innerHTML = myXMLHttp.responseText;
//This will write the responseText value to the HTML Element with attribute id=”myId”
        }
};
myXMLHttp.open("GET", "ajaxtest.txt", true);
myXMLHttp.setRequestHeader("X-Requested-With", "XMLHttpRequest"); //This tells the web server that the call is made for ajax purposes
myXMLHttp.send(null); //No data need to send along with the request
```

In the above example, we have used the “*responseText”* property to get the “*Plain Text*” response from the web server. We can also use the “*responseXML”* property to read our XML file “myxmlfile.xml” that we had created earlier in this Unit. This is shown in the example below.

Example:

```
if (window.XMLHttpRequest)
{
	var myXMLHttp = new XMLHttpRequest();
}
else
{
	throw new Error("Your Browser does not support the XMLHttpRequest object. Please upgrade your browser.");
}

myXMLHttp.onreadystatechange = function() {
	if (myXMLHttp.readyState === 4){
		var output, parent, child;
		output = "<table><tr><th>Book Title</th><th>Author</th><th>Publisher</th><th>Year</th></tr>";
		parent = myXMLHttp.responseXML.documentElement.getElementsByTagName("book");
		for ( i=0; i<parent.length; i++ )
		{
			output = output + "<tr>";
			child = parent[i].getElementsByTagName("title");
			{
			try
			{
				output = output + "<td>" + child[0].firstChild.nodeValue + "</td>";
			}
			catch (er)
			{
				output = output + "<td> </td>";
			}
		}
		child = parent[i].getElementsByTagName("author");
```

```
		{
			try
			{
				output = output + "<td>" + child[0].firstChild.nodeValue + "</td>";
			}
			catch (er)
			{
				output = output + "<td> </td>";
			}
		}
		child = parent[i].getElementsByTagName("publisher");
		{
			try
			{
				output = output + "<td>" + child[0].firstChild.nodeValue + "</td>";
			}
			catch (er)
			{
				output = output + "<td> </td>";
			}
		}
		child = parent[i].getElementsByTagName("year");
		{
			try
			{
				output = output + "<td>" + child[0].firstChild.nodeValue + "</td>";
			}
			catch (er)
			{
				output = output + "<td> </td>";
			}
```

```
                }
                output = output + "</tr>";
        }
        output = output + "</table>";
        document.getElementById(myId).innerHTML = output;
        }
};
myXMLHttp.open("GET", "myxmlfile.xml", true);
myXMLHttp.setRequestHeader("X-Requested-With", "XMLHttpRequest");   //This
tells the web server that the call is made for ajax purposes
myXMLHttp.send(null);  //No data need to send along with the request
```

5.5 Applying What We Have Learnt VII – Adding Ajax Functionality in JavaScript

We already have a JavaScript file named “myjavascripts.js”, from our previous unit. To add the Ajax functionality to our JavaScript file, open it using a text editor and add to it the following content, shown below. Though you could use other advanced editors, it is recommended that you use a simple text editor to get your concepts clear.

myjavascripts.js

```
//Function that takes the "id" attribute of an HTML element as its parameter
// to display the Text responses in it.
function myAjaxShowText(myId)
{
        if (window.XMLHttpRequest)
        {
                var myXMLHttp = new XMLHttpRequest();
        }
        else
        {
```

```
            throw new Error("Your Browser does not support the XMLHttpRequest object. Please upgrade your browser.");
        }

        myXMLHttp.onreadystatechange = function() {
            if (myXMLHttp.readyState === 4){
                //This will write the responseText value to the HTML Element with attribute id=”myId”
                document.getElementById(myId).innerHTML = myXMLHttp.responseText;
            }
        };
        myXMLHttp.open("GET", "ajaxtest.txt", true);
        //This tells the web server that the call is made for ajax purposes
        myXMLHttp.setRequestHeader("X-Requested-With", "XMLHttpRequest");
        //No data need to send along with the request
        myXMLHttp.send(null);
}

//Function that takes the "id" attribute of an HTML element as its parameter
// to display the XML responses in it.
function myAjaxShowXML(myId)
{
        if (window.XMLHttpRequest)
        {
            var myXMLHttp = new XMLHttpRequest();
        }
        else
        {
            throw new Error("Your Browser does not support the XMLHttpRequest object. Please upgrade your browser.");
        }

        myXMLHttp.onreadystatechange = function() {
```

```
            if (myXMLHttp.readyState === 4){

                var output, parent, child;

                output = "<table border='1' cellpadding='5' cellspacing='0'><tr><th>Book Title</th><th>Author</th><th>Publisher</th><th>Year</th></tr>";
                parent = myXMLHttp.responseXML.documentElement.getElementsByTagName("book");
                for ( i=0; i<parent.length; i++ )
                {
                    output = output + "<tr>";
                    child = parent[i].getElementsByTagName("title");
                    {
                        try
                        {
                            output = output + "<td>" + child[0].firstChild.nodeValue + "</td>";
                        }
                        catch (er)
                        {
                            output = output + "<td> </td>";
                        }
                    }

                    child = parent[i].getElementsByTagName("author");
                    {
                        try
                        {
                            output = output + "<td>" + child[0].firstChild.nodeValue + "</td>";
                        }
                        catch (er)
                        {
```

```
                                        output = output + "<td> </td>";
                                }
                        }

                        child                                                   =
parent[i].getElementsByTagName("publisher");
                        {
                                try
                                {
                                        output    =    output    +    "<td>"    +
child[0].firstChild.nodeValue + "</td>";
                                }
                                catch (er)
                                {
                                        output = output + "<td> </td>";
                                }
                        }

                        child = parent[i].getElementsByTagName("year");
                        {
                                try
                                {
                                        output    =    output    +    "<td>"    +
child[0].firstChild.nodeValue + "</td>";
                                }
                                catch (er)
                                {
                                        output = output + "<td> </td>";
                                }
                        }

                        output = output + "</tr>";
                }
                output = output + "</table>";
```

```
                document.getElementById(myId).innerHTML = output;
            }
        };
        myXMLHttp.open("GET", "myxmlfile.xml", true);
        //This tells the web server that the call is made for ajax purposes
        myXMLHttp.setRequestHeader("X-Requested-With", "XMLHttpRequest");
        //No data need to send along with the request
        myXMLHttp.send(null);
}
```

We have created a JavaScript file that has some Ajax functionality which will be used in the next section to add some dynamism to the login form we had modified previously in Unit 4, “Applying What We Have Learnt - VI”.

5.6 Applying What We Have Learnt VIII – Adding Ajax Functionality to a Web Page

Now, we add the “myjavascripts.js” file we have modified in the previous section to the “index.html” file that we had modified in the previous Unit 4, “Applying What We Have Learnt – VI” section, as shown below. Notice that we have added few more elements to our “index.html”, marked in bold.

Putting it all together, the list of files with contents:

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <noscript>JavaScript support is required to view the contents of this page. Please enable JavaScript or use a browser that supports it. Thank you!</noscript>
        <script src="myjavascripts.js" type="text/javascript"></script>
```

```
        <link rel="stylesheet" href="mystyle.css" type="text/css">
        <title>My Website</title>
    </head>
    <body>
        <p>Welcome to my Website</p>
        <p id="response1" onclick="myAjaxShowXML('response1')" onmouseout="myAjaxHideXML('response1')">View My XML File</p>
        <form action="process.php" method="post">
        <table class="login-form">
            <tr>
                <td>Message:</td><td><input id="messageBox" type="text" name="messageBox" readonly></td>
            </tr>
            <tr>
                <td>Username:</td><td><input id="username" type="text" name="username" onkeyup="checkUsername()"></td>
            </tr>
            <tr>
                <td>Password:</td><td><input id="password" type="password" name="password"></td>
            </tr>
            <tr>
                <td colspan="2"><input id="login" type="submit" value="Login"></td>
            </tr>
        </table>
        </form>
        <div id="response2" onmouseover="myAjaxShowText('response2')" onmouseout="myAjaxHideText('response2')">Put the mouse pointer over me :-)</div>
    </body>
</html>
```

The “myjavascripts.js” file should now have the contents as shown below.

myjavascripts.js

```
//function checkUsername checks the username input; whether blank,
//numbers, letters or special characters.
function checkUsername()
{
	var username;
	username = document.getElementById("username").value;

	if ( username=="" )
	{
		document.getElementById("messageBox").value = "Blank
Username";
	}
	else if ( username.match(/[a-z]/i) )
	{
		document.getElementById("messageBox").value = "Alphabets";
	}
	else if ( username.match(/[0-9]/) )
	{
		document.getElementById("messageBox").value = "Numbers";
	}
	else
	{
		document.getElementById("messageBox").value = "Special
Characters";
	}
}

//Function that takes the "id" attribute of an HTML element as its parameter
// to display the Text responses in it.
function myAjaxShowText(myId)
{
```

```
	if (window.XMLHttpRequest)
	{
		var myXMLHttp = new XMLHttpRequest();
	}
	else
	{
		throw new Error("Your Browser does not support the XMLHttpRequest object. Please upgrade your browser.");
	}

	myXMLHttp.onreadystatechange = function() {
		if (myXMLHttp.readyState === 4){
			//This will write the responseText value to the HTML Element with attribute id=myId
			document.getElementById(myId).innerHTML = myXMLHttp.responseText;
		}
	};
	myXMLHttp.open("GET", "ajaxtest.txt", true);
	//This tells the web server that the call is made for ajax purposes
	myXMLHttp.setRequestHeader("X-Requested-With", "XMLHttpRequest");
	//No data need to send along with the request
	myXMLHttp.send(null);
}

//Function that takes the "id" attribute of an HTML element as its parameter
// to display the XML responses in it.
function myAjaxShowXML(myId)
{
	if (window.XMLHttpRequest)
	{
		var myXMLHttp = new XMLHttpRequest();
	}
	else
```

```
        {
                throw new Error("Your Browser does not support the XMLHttpRequest object. Please upgrade your browser.");
        }

        myXMLHttp.onreadystatechange = function() {
                if (myXMLHttp.readyState === 4){

                        var output, parent, child;
                        output = "<table border='1' cellpadding='5' cellspacing='0'><tr><th>Book Title</th><th>Author</th><th>Publisher</th><th>Year</th></tr>";
                        parent = myXMLHttp.responseXML.documentElement.getElementsByTagName("book");
                        for ( i=0; i<parent.length; i++ )
                        {
                                output = output + "<tr>";
                                child = parent[i].getElementsByTagName("title");
                                {
                                        try
                                        {
                                                output = output + "<td>" + child[0].firstChild.nodeValue + "</td>";
                                        }
                                        catch (er)
                                        {
                                                output = output + "<td> </td>";
                                        }
                                }

                                child = parent[i].getElementsByTagName("author");
                                {
                                        try
                                        {
```

```
                                        output = output + "<td>" +
child[0].firstChild.nodeValue + "</td>";
                                }
                                catch (er)
                                {
                                        output = output + "<td> </td>";
                                }
                        }

                        child                                           =
parent[i].getElementsByTagName("publisher");
                        {
                                try
                                {
                                        output = output + "<td>" +
child[0].firstChild.nodeValue + "</td>";
                                }
                                catch (er)
                                {
                                        output = output + "<td> </td>";
                                }
                        }

                        child = parent[i].getElementsByTagName("year");
                        {
                                try
                                {
                                        output = output + "<td>" +
child[0].firstChild.nodeValue + "</td>";
                                }
                                catch (er)
                                {
                                        output = output + "<td> </td>";
                                }
```

```
                                }

                                output = output + "</tr>";
                        }
                        output = output + "</table>";
                        document.getElementById(myId).innerHTML = output;
                }
        };
        myXMLHttp.open("GET", "myxmlfile.xml", true);
        //This tells the web server that the call is made for ajax purposes
        myXMLHttp.setRequestHeader("X-Requested-With", "XMLHttpRequest");
        //No data need to send along with the request
        myXMLHttp.send(null);
}

function myAjaxHideText(myId)
{
        document.getElementById(myId).innerHTML = "Put the mouse pointer over me :-)";
}

function myAjaxHideXML(myId)
{
        document.getElementById(myId).innerHTML = "View My XML File";
}
```

The “mystyle.css” file has the contents as shown below.

mystyle.css

```
/* login-form class definition */
.login-form {
        color:darkred; /* darkred text colour */
        text-transform:uppercase; /* text in uppercase */
        border-style:dotted; /* dotted border */
```

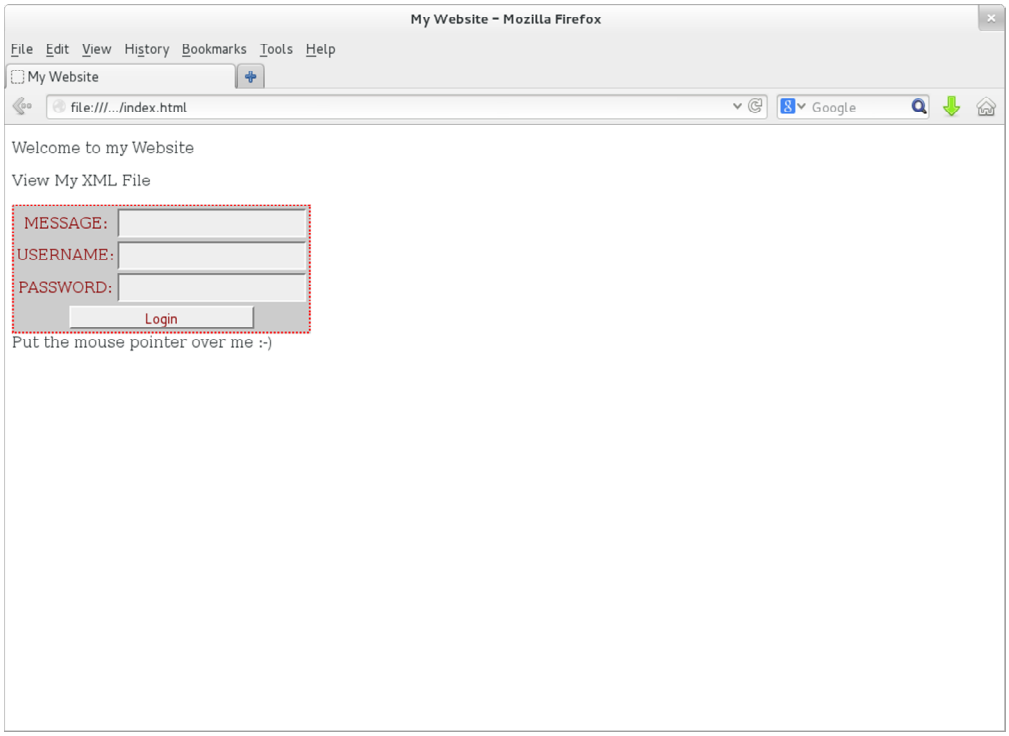
```
        border-width:2px; /* 2 pixels border width */
        border-color:#ff0000;  /* red border colour */
        background-color:#cccccc; /* background colour is a shade of grey */
}

/* apply to all the td elements of the login-form class */
.login-form td {
    text-align:center; /* center align td elements */
    vertical-align:middle;  /* vertically middle align td elements */
}

/* apply to all the input elements of the login-form class */
.login-form input {
        color:darkred; /* input elements have darkred text colour */
        background-color:#eeeeee; /* input elements have grey background */
        height:25px; /* input elements have 25 pixels height */
        width:200px; /* input elements have 200 pixels width */
}
```

The “ajaxtext.txt” file has the contents as shown below.

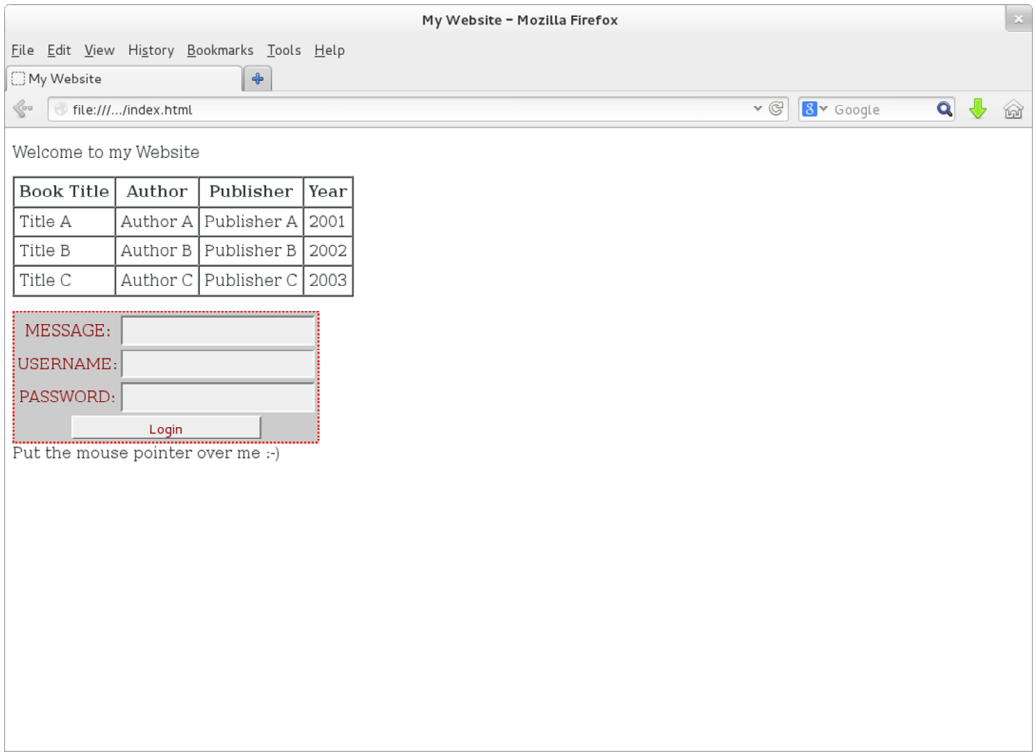
ajaxtest.txt

```
If you can read this, Ajax is working. This content was accessed using the "responseText" property of the XMLHttpRequest object.
```

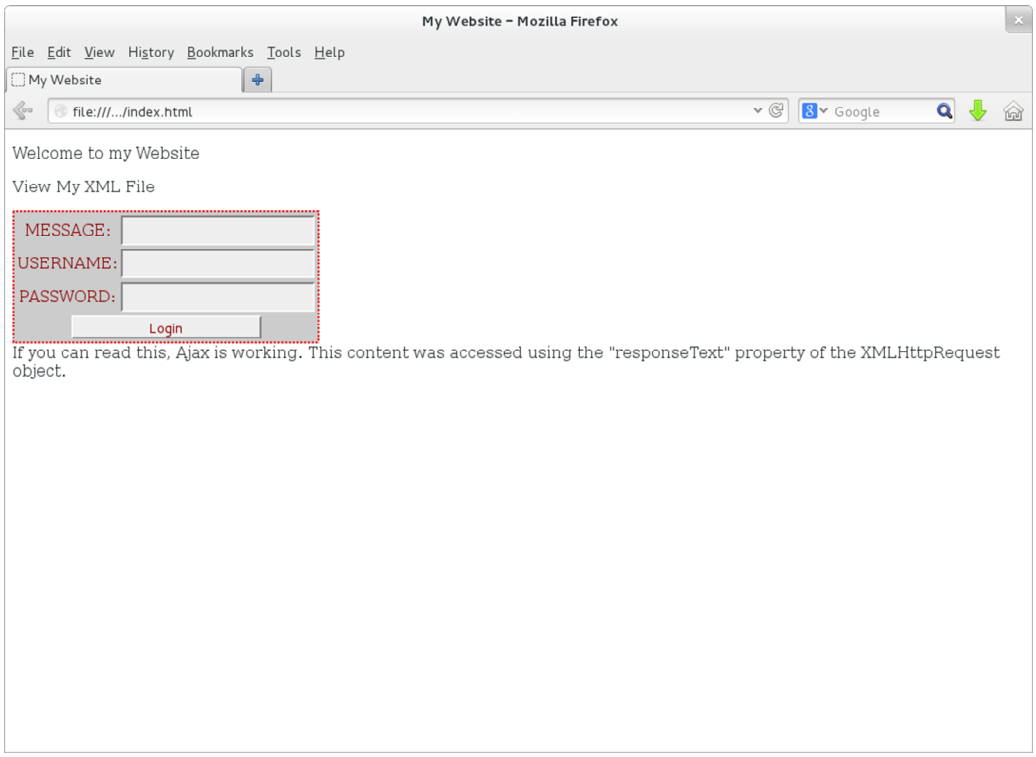
Once you have all these above mentioned list of files in a folder, open the “index.html” file in your browser. You should see a similar page as shown in the picture below.



If you click (or double-click) on the “View My XML File” you will observe Ajax in action as shown in the picture below. In this example, we use Ajax to read from an XML file. As soon as you navigate the mouse pointer away from the text, the table disappears.



Also, you can observe that every time you navigate your mouse pointer over the text that reads “Put the mouse pointer over me :-)”, the text changes to the text as shown in the picture below.



In both the examples depicted above, the entire HTML page is not loaded only the respective “ ... ” and “ ... ” HTML elements are updated. This is a tremendous improvement and also provides some dynamism to our “index.html” page. This example should intrigue you to further explore the possibilities of dynamism that can be applied to web pages using Ajax. We have not covered all the topics extensively however, the purpose of this unit was to expose you to the basics of both XML and Ajax and using them on web pages.

5.7 LET US SUM UP

In this unit, we have briefly discussed the basics of XML and Ajax. Though we have covered only a few of both the topics, these topics are intended to inspire and point you in a direction to further explore. This unit was not intended to provide you with an exhaustive in-depth on the topics but merely to introduce you to some of the basic concepts of XML and Ajax and use them in web pages.

What we have learned in this unit.

- XML and Ajax basics
- Creating an XML file
- Adding Ajax functionality to JavaScript
- Using XML and Ajax in a Web page

CHECK YOUR PROGRESS

Q1: XML is

(a) a web page (b) a scripting language

(c) a css file (d) a markup language

Q2: Which of the following file extension(s) is(are) used for an XML file?

(a) .html (b) .css

(c) .xml (d) .js

Q3: Which of the following is a correct syntax for enclosing comments in XML?

(a) <! this is a HTML comment --> (b) /* this is a XML comment */

(c) // this is a JavaScript comment (d) this is a JavaScript comment //

Q4: For an XML declaration, which of the following tags are used?

(a) “<script> … </script>” tags (b) ”<?xml … ?>” tags

(c) “<noscript> … </noscript>” tags (d) “<xml></xml>” tags

Q5: In XML, which of the following character(s) have to be replaced with the entity reference “&” ?

(a) ' (b) amp

(c) & (d) <

Q6: Which browser API can be used to send HTTP/HTTPS requests to a web server and load the server response data back into the script asynchronously?

Q7: Which of the following are methods of the XMLHttpRequest object, that are used to send requests to the web server?

(a) readyState (b) open

(c) onreadystatechange (d) send

Q8: Once the HTTP response content has finished loading, which of the following is assigned a value of “4”?

(a) readyState (b) onreadystatechange

(c) open (d) send

Q9: To submit HTML form data, which XMLHttpRequest object's method is needed, if the HTTP method specified in the *open()* method is POST?

(a) setRequestHeader (b) open

(c) send (d) readyState

Q10: Which of the following XMLHttpRequest property can be used to read XML files?

(a) responseXML (b) responseText

(c) readyState (d) None of the above.

5.8 FURTHER READINGS

- Michael Morrison, "Sams Teach Yourself XML in 24 Hours Complete Starter Kit 3rd Edition", Dorling Kindersley, ISBN-13 9788131700129, ISBN-10 8131700127, 2006
- Steven Holzner, "Ajax Bible 1st Edition", Wiley India Pvt Ltd, ISBN-13 9788126512171, ISBN-10 8126512172, 2009
- https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started
- http://www.w3.org/XML/
- http://www.w3.org/TR/XMLHttpRequest/

5.9 ANSWERS TO CHECK YOUR PROGRESS

1. (d)

2. (c)

3. (a)

4. (b)

5. (c)

6. XMLHttpRequest Application Programming Interface.

7. (b), (d)

8. (a)

9. (a)

10. (a)

5.10 MODEL QUESTIONS

1. What is XML and what is it used for?
2. Which file extension is used to store Ajax code?
3. How do we test browser support for the **XMLHttpRequest** object?
4. Using the *open()* method if we need to send data to the web server, which HTTP method can be used?
5. In XML, what are *sibling* elements?
6. What is the **onreadystatechange** event listener?
7. Which pair of tags are used to enclose comments in XML?
8. What is **responseText**?
9. What is **responseXML**?
10. What is a Ajax?

****

UNIT 6: PHP

UNIT STRUCTURE

6.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the basics of PHP
- write simple PHP code

- use PHP in web pages

6.2 INTRODUCTION

PHP is a server-side scripting language designed for web development and also used as a general-purpose programming language. JavaScript on the other hand is a client-side scripting language. PHP was created by Rasmus Lerdorf in 1995. Currently the PHP Group produces the reference implementation of PHP. PHP was originally an acronym for Personal Home Page, however it now stands for the recursive acronym "**P**HP: **H**ypertext **P**reprocessor".

The PHP code on a web server is interpreted by the web server using a PHP processor module. Though the PHP processor module can generate data in formats like JSON, XML, etc., it commonly is used to generate web pages in HTML. Furthermore, PHP commands can be embedded directly into an HTML document. PHP is a free software *(free as in Freedom and not the money value),* released under the PHP License and is deployed on most web servers today. This unit aims to provide the basic understanding of PHP to build web pages.

We will be using **Mozilla Firefox** (version 27+) as our browser to test all our PHP codes. Since, PHP scripts have to run on a web server there are some additional topics we will need to cover in this unit – namely the **Apache Web Server**. We will need this to test the PHP code that will be written in the unit examples. Also, we will cover the **MySQL Database Server** briefly so as to be able to test our PHP codes for database access.

6.3 SOFTWARE PREREQUISITES

PHP is mainly focused on server-side scripting, therefore it can be used to collect form data, generate dynamic page content, or send and receive cookies. However, there are three main areas where PHP scripts can be used.

1) *Server-side scripting*

To use PHP for server-side scripting we need the PHP parser, a web server and a web browser. You need to run the web server with a connected PHP installation. You can access the PHP program output on a web browser, viewing the PHP page from the web server. This usage area of PHP is what we will be covering in this unit.

2) *Command line scripting*

PHP also supports a CLI SAPI as of PHP v4.3.0. The main focus of this SAPI is for developing shell applications with PHP. This provides us with the option to run PHP scripts without any web server or web browser, requiring only the PHP parser.

3) *Writing desktop applications*

PHP is probably not the appropriate language to create a desktop applications with a GUI, but if you know PHP very well, and would like to use some advanced PHP features in your client-side applications you can use the PHP-GTK to write such programs. You also have the ability to write cross-platform applications this way. PHP-GTK is an extension to PHP, not available in the main distribution though.

Our first prerequisite to running service-side PHP scripts is a web server and we choose the Apache Web Server. The second prerequisite is PHP and the third prerequisite MySQL. Now, depending upon your Operating System you can choose either LAMP (Linux Apache MySQL PHP) or WAMP (Windows Apache MySQL PHP) installation. MAMP (**M**y **A**pache **M**ySQL **P**HP) can be used by Mac OS users.

For Windows and Mac OS installations, you have the option to install available software like **XAMPP** *(available for download at http://www.apachefriends.org/index.html)*, **WAMP** *(available for download at http://www.wampserver.com/en/)*, **MAMP** *(available for download at http://www.mamp.info/en/)* to name a couple among a host of others. Either one of these will do, just ensure that you have a working Apache, MySQL and PHP

running on your computer before proceeding to the remaining sections of this unit.

For Linux installations, you may already have the Apache, MySQL and PHP installed on your system. Nonetheless, we will go ahead and discuss the installation steps on the Fedora distribution of Linux, as an example. The installation steps on other distributions of Linux should be fairly similar.

6.3.1 Installing Apache and PHP on Fedora 18

[fedorauser@kkhsou ~]$ **su -c "/usr/bin/yum groupinstall 'Web Server'"**
The command shown above requires you to provide the "root" password for its successful execution.

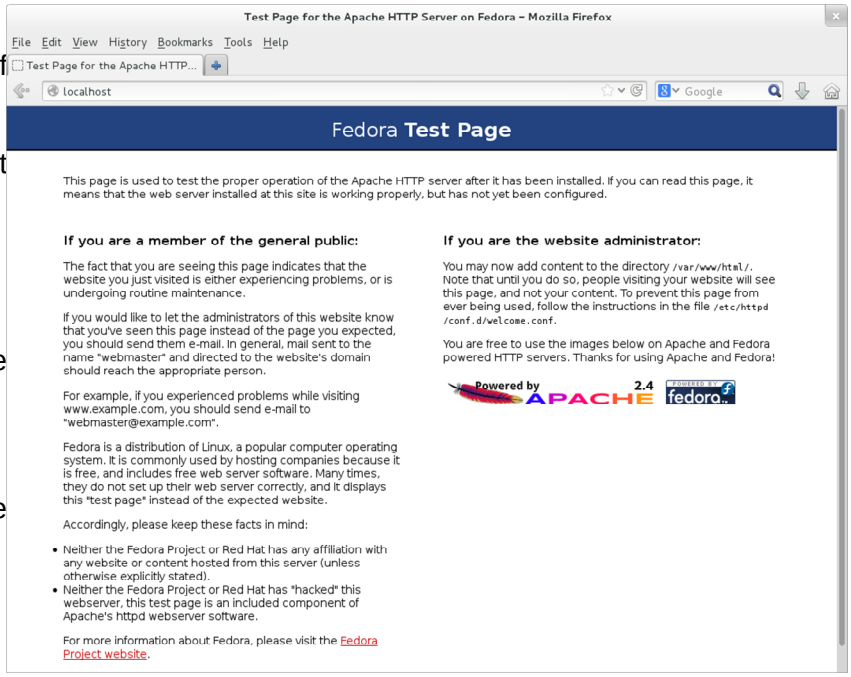
The command above installs the software needed to run Apache with the support for database driven web sites, support for common web scripting languages, such as PHP, perl, and python, Apache documentation provided by httpd-manual rpm package and support for serving secure, encrypted content through HTTPS protocol.

6.3.2 Starting and Testing Apache on Fedora 18

[fedorauser@kkhsou ~]$ **su -c "systemctl start httpd.service"**
This command shown above requires you to provide the "root" password for its successful execution.

The command above starts the Apache Web Server service. To test the correct operation of the Apache server, open the link "http://localhost" in your web browser. If the browser displays the "Fedora Test Page", as shown in the picture below, Apache is installed and working correctly.

List of directories that contain files needed for the proper operation of the Apache web server.



/etc/httpd : The location of **Apache configuration files**, referred to as ServerRoot.

/usr/lib/httpd/modules : The location of various **Apache modules**, loaded on demand from the main configuration file.

/var/www/html : Default location for storing web site content, referred to as **DocumentRoot**.

/var/log/httpd : The location of the **Apache log files**.

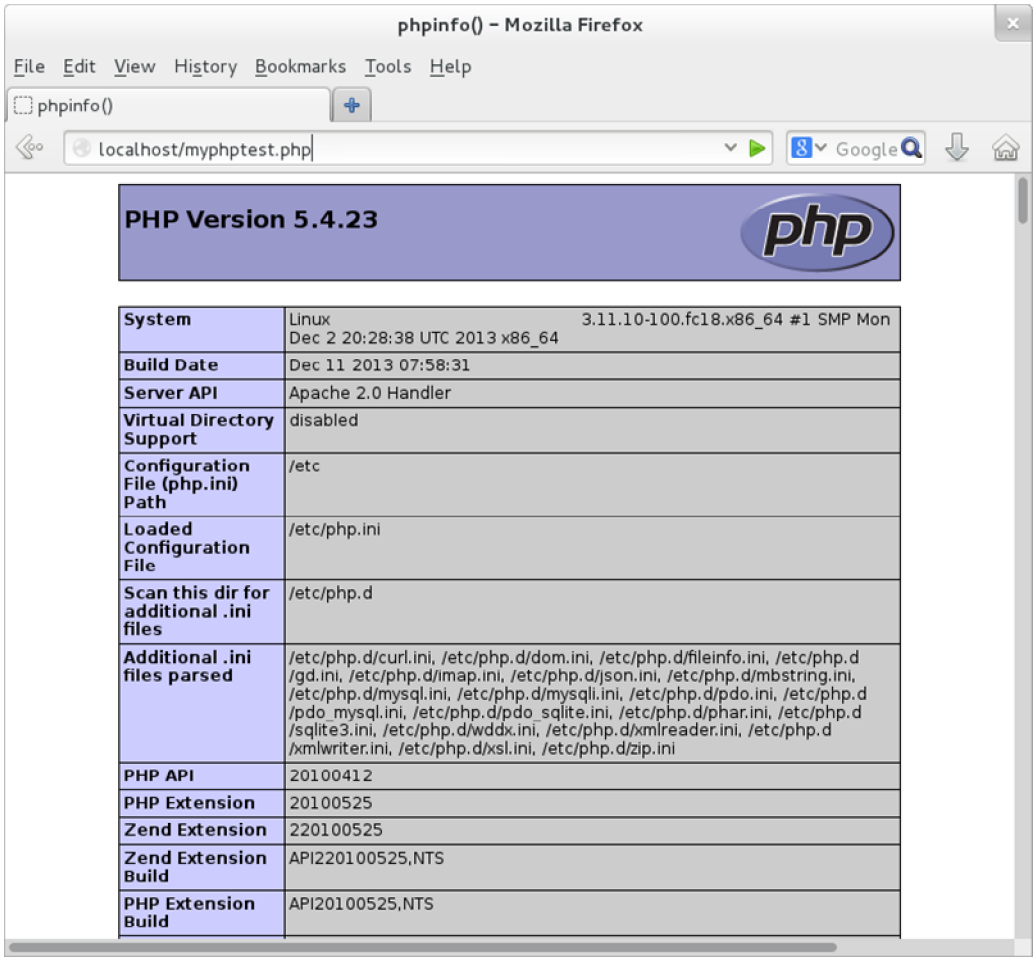
6.3.3 Testing PHP with phpinfo()

To test if PHP is installed and working properly, we will create a PHP file called “myphptest.php” in the **DocumentRoot** folder and populate it with the content shown below.

myphptest.php

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

Opening the link “http://localhost/myphptest.php” in the web browser should display a page similar to the one shown below.



System	Linux 3.11.10-100.fc18.x86_64 #1 SMP Mon Dec 2 20:28:38 UTC 2013 x86_64
Build Date	Dec 11 2013 07:58:31
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/curl.ini, /etc/php.d/dom.ini, /etc/php.d/fileinfo.ini, /etc/php.d/gd.ini, /etc/php.d/imap.ini, /etc/php.d/json.ini, /etc/php.d/mbstring.ini, /etc/php.d/mysql.ini, /etc/php.d/mysqli.ini, /etc/php.d/pdo.ini, /etc/php.d/pdo_mysql.ini, /etc/php.d/pdo_sqlite.ini, /etc/php.d/phar.ini, /etc/php.d/sqlite3.ini, /etc/php.d/wddx.ini, /etc/php.d/xmlreader.ini, /etc/php.d/xmlwriter.ini, /etc/php.d/xsl.ini, /etc/php.d/zip.ini
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525
Zend Extension Build	API220100525,NTS
PHP Extension Build	API20100525,NTS

6.3.4 Installing MySQL on Fedora 18

```
[fedorauser@kkhsou ~]$ su -c "yum install mysql-server"
```

This command shown above requires you to provide the “root” password for its successful execution.

The command shown above, installs the packages for MySQL server including

packages for the MySQL client.

6.3.5 Starting and Testing MySQL on Fedora 18

```
[fedorauser@kkhsou ~]$ su -c "systemctl start mysqld.service"
```

This command shown above requires you to provide the "root" password for its successful execution.

The command above starts the MySQL Server service. To test whether we are able to connect to the MySQL Server, we can use the MySQL client available in the command line interface on Linux, as shown below.

```
[fedorauser@kkhsou ~]$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.5.35 MySQL Community Server (GPL)
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

If you happen to see a "**mysql>**" prompt, as shown above then the MySQL Server is installed and working correctly. Do note here, that by default the mysql root account does not have a password initially, we need to create a password for this account, if required. However, for our purposes we will go ahead with all the defaults.

6.3.6 Installing the PHP-MySQL module

We are almost done with our prerequisites. Just one more module named

“php-mysql” will be required by PHP applications that use MySQL databases. To install this module, we can use the command shown below.

```
[fedorauser@kkhsou ~]$ su -c "yum install php-mysql"
```

This command shown above requires you to provide the “root” password for its successful execution.

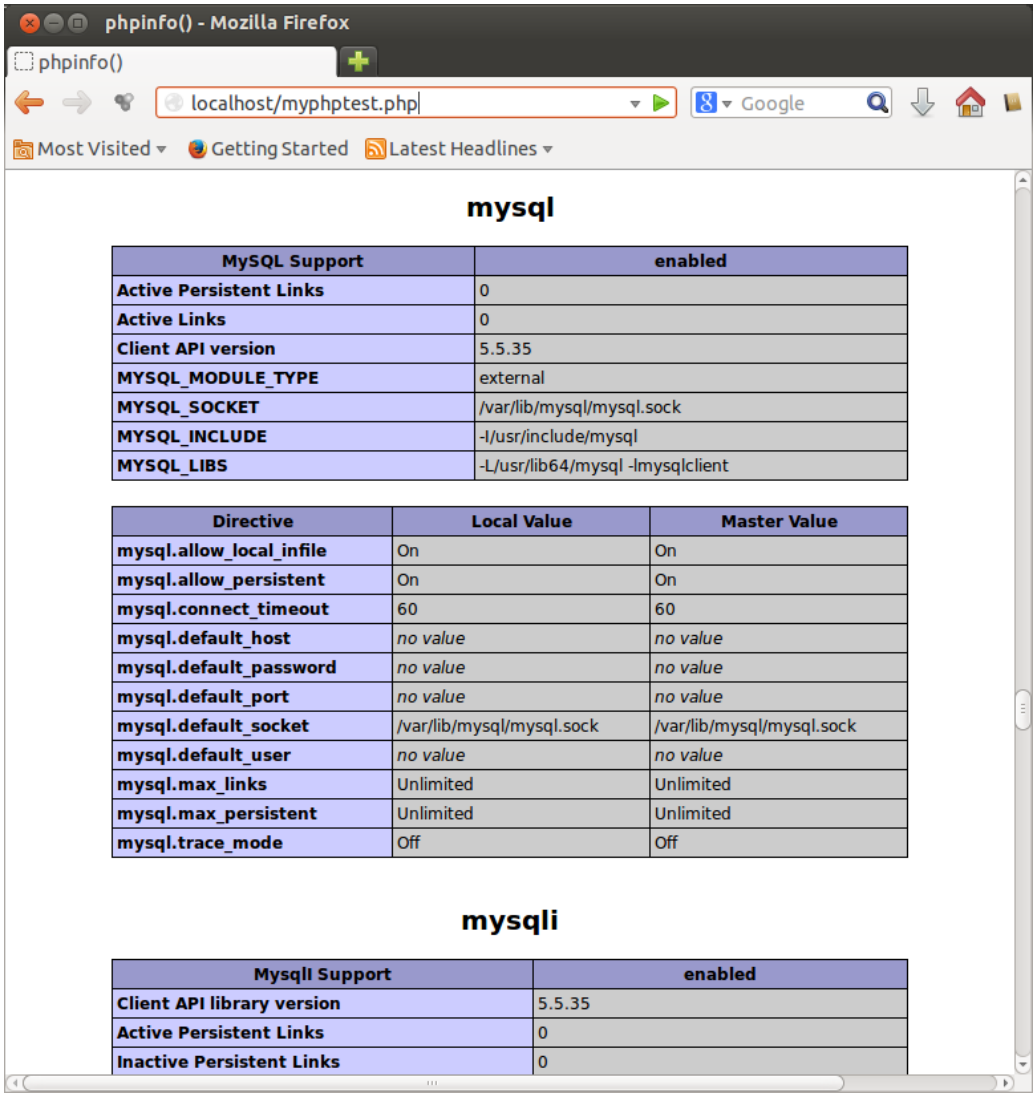
After the installation of this module, we will need to reload the Apache Web Server to include this module. This can be done using the command shown below.

```
[fedorauser@kkhsou ~]$ su -c "systemctl reload httpd.service"
```

This command shown above requires you to provide the “root” password for its successful execution.

6.3.7 Checking the PHP-MySQL module

Once we have reloaded the Apache Web Server, we can test our PHP information by opening the link “http://localhost/myphptest.php” in the web browser, which should now display a page similar to the one shown below indicating that the **php-mysql** module is loaded and available. Just look for “mysql” and “mysqli” on this page. **mysqli** stands for the MySQL Improved Extension.



mysql

MySQL Support	enabled
Active Persistent Links	0
Active Links	0
Client API version	5.5.35
MYSQL_MODULE_TYPE	external
MYSQL_SOCKET	/var/lib/mysql/mysql.sock
MYSQL_INCLUDE	-I/usr/include/mysql
MYSQL_LIBS	-L/usr/lib64/mysql -lmysqlclient

Directive	Local Value	Master Value
mysql.allow_local_infile	On	On
mysql.allow_persistent	On	On
mysql.connect_timeout	60	60
mysql.default_host	*no value*	*no value*
mysql.default_password	*no value*	*no value*
mysql.default_port	*no value*	*no value*
mysql.default_socket	/var/lib/mysql/mysql.sock	/var/lib/mysql/mysql.sock
mysql.default_user	*no value*	*no value*
mysql.max_links	Unlimited	Unlimited
mysql.max_persistent	Unlimited	Unlimited
mysql.trace_mode	Off	Off

mysqli

Mysqli Support	enabled
Client API library version	5.5.35
Active Persistent Links	0
Inactive Persistent Links	0

At this stage, we have installed all the software prerequisites required by the PHP code examples of this unit. The PHP code examples were tested in the following version of the prerequisites. Apache v2.4.6, PHP v5.4.23 and MySQL v5.5.35.

6.4 GETTING STARTED WITH PHP

PHP code is executed on the server-side unlike JavaScript which is client-side. PHP code is generally stored in files with the file extension ".php". You should use a simple text editor to write all of the PHP code provided in the

examples. This will keep your concepts clear.

When the PHP parser parses a file, it looks for the opening "<?php" and closing “?>” PHP tags which tell the PHP parser to respectively start and stop interpreting the code between the tags. Parsing in this manner by the PHP parser, allows PHP code to be embedded in different types of documents, as everything outside the pair of opening and closing PHP tags is ignored by the PHP parser.

6.4.1 Basic PHP Syntax

PHP Tags

PHP code is enclosed within a start "<?php" and an end "?>" processing tag as shown below.

```
<?php
        … PHP code goes here ...
?>
```

Comments

PHP supports 'C', 'C++' and Unix shell-style (Perl style) comments. Comments in PHP code can be used as shown below.

```
<?php
        // This is a one line comment in PHP
        /* This is a multi
           line comment in PHP */
        # This is another one line comment in PHP
?>
```

Instruction separation

As in C or Perl, PHP requires instructions to be terminated with a semicolon “;” at the end of each statement.

```
<?php
echo "Hi I am a PHP output";
?>
```

6.4.2 Data Types

PHP supports eight primitive data types. Four scalar data types - boolean, integer, float (double), string. Two compound data types - array, object. Two special data types - resource, NULL. The data type of a variable is not usually set by the programmer but at runtime by the PHP parser depending on the context in which that variable is used.

boolean

This is the simplest type. A boolean expresses a truth value. It can be either TRUE or FALSE.

Syntax:

To specify a boolean literal, use the keywords TRUE or FALSE. Both are case-insensitive.

Example:

```
<?php
//This assigns the value TRUE to the variable $my_boolean
$my_boolean = True;
?>
```

integer

An integer is a number of the set ℤ = {..., -2, -1, 0, 1, 2, ...}.

Syntax:

Integers can be specified in decimal notation (base 10, by specifying the number), octal notation (base 8, by preceding the number with a "0"), hexadecimal notation (base 16, by preceding the number with a "0x"), binary notation (base 2, by preceding the number with a "0b"), optionally preceded by a sign (- or +).

Example:

```
<?php
//Assigns a positive decimal number to the variable $my_integer
$my_integer = 31;

//Assigns a negative decimal number to the variable $my_integer
$my_integer = -5;

//Assigns an octal number to the variable $my_integer
$my_integer = 0123;

//Assigns a hexadecimal number to the variable $my_integer
$my_integer = 0x1A;

//Assigns binary number to the variable $my_integer
$my_integer = 0b11111111;
?>
```

float (double)

A float is used to represent the rational numbers such as the integer −5 and the fraction 4/3, and the irrational numbers such as √2 (1.41421356… the square root of two, an irrational algebraic number) and pi (3.14159265…, a transcendental number).

Syntax:

Floating point numbers are also known as "floats", "doubles", or "real numbers" and can be specified using any of the following syntax.

Example:

```
<?php
//This assigns a real number to the variable $my_float
$my_float = 1.234;

//This assigns a real number to the variable $my_float
$my_float = 0.123;
```

```
?>
```

string

A string is series of characters, where a character is the same as a byte.

single quoted syntax - The simplest way to specify a string is to enclose within single quotes ' '.

Example:

```
<?php
//Assigns a string to the variable $my_string
$my_string = 'My single quoted string';
?>
```

double quoted syntax - A string can also be specified enclosed within double-quotes " ".

Example:

```
<?php
//Assigns a string to variable $my_string
$my_string = “My double quoted string”;
?>
```

heredoc syntax – In this syntax, after the "<<<" operator, an identifier is provided, then a newline, the string itself follows, and thereafter the same identifier again with a semicolon ";" to close the quotation. The closing identifier must begin in the first column of the line. Also, the identifier must follow the same naming rules as any other label in PHP i.e., it must contain only alphanumeric characters and underscores, and must start with a non-digit character or underscore.

Example:

```
<?php
$my_string = <<<MY_IDENTIFIER
This is an example of a string
```

```
spanning multiple lines
using the heredoc syntax.
MY_IDENTIFIER;
?>
```

The above example can also be written as shown below.

```
<?php
$my_string = <<<"MY_IDENTIFIER"
This is an example of a string
spanning multiple lines
using the heredoc syntax.
MY_IDENTIFIER;
?>
```

nowdoc syntax - nowdocs are to single-quoted strings what heredocs are to double-quoted strings. A nowdoc is specified similarly to a heredoc, but no parsing is done inside a nowdoc. The construct is ideal for embedding PHP code or other large blocks of text without the need for escaping. It declares a block of text which is not for parsing. A nowdoc is identified with the same <<< sequence used for heredocs, but the identifier which follows is enclosed in single quotes, e.g. <<<'MY_IDENTIFIER'. All the rules for heredoc identifiers also apply to nowdoc identifiers, especially those regarding the appearance of the closing identifier.

Example:

```
<?php
$my_string = <<<'MY_IDENTIFIER'
This is an example of a string
spanning multiple lines
using the nowdoc syntax.
MY_IDENTIFIER;
?>
```

array

An array in PHP is actually an ordered map. A map is a data type that associates values to keys. This data type is optimized for several different uses; it can be treated as an array, list (vector), hash table (an implementation of a map), dictionary, collection, stack, queue, etc.

Syntax:

An array can be created using the *array()* PHP construct. It takes any number of comma-separated "**key => value**" pairs as arguments. The "**key**" can either be an integer or a string. The "**value**" can be of any data type.

Example:

```
<?php
$my_array = [
    "1" => "K",
    "2" => "K",
    "3" => "H",
    "4" => "S",
    "5" => "O",
    "6" => "U",
];
?>
```

The *unset()* PHP function can be used to remove elements or delete an array, as shown below.

```
<?php
unset($my_array[5]); // This removes "O" from the array
unset($my_array);    // This deletes the whole array
?>
```

object

To use a class which provides the definitions of properties and methods belonging to the class, it has to be instantiated as an object.

Syntax:

An object is created using the "new" statement to instantiate a class.

Example:

```
<?php
// Class definition
class my_class
{
   function my_method()
   {
      echo "I am a text within my_method()";
   }
}

// An object of my_class instantiated
$my_object = new my_class;

// A method of my_class called via the object of the class
$my_object->my_method();
?>
```

resource

A resource is a special variable, holding a reference to an external resource like handlers to opened files, database connections, image canvas areas, etc. Resources are created and used by special functions.

NULL

The special NULL value represents a variable with no value. NULL is the only possible value of type null.

Syntax:

There is only one value of type null, and that is the case-insensitive constant NULL.

Example:

```
<?php
$my_variable = NULL;
?>
```

6.4.3 Variables

The variables in PHP are represented by the dollar "$" sign followed by the name of the variable. The variable name is case-sensitive. Variable names can start with a letter or underscore, followed by any number of letters, numbers, or underscores.

By default, variables are always "assigned by value" i.e., when you assign an expression to a variable, the entire value of the original expression is copied into the destination variable. Therefore, after assigning one variable's value to another, changing one of those variables will have no effect on the other.

Example:

```
<?php
$my_variable = 'KKHSOU';
$my_Variable = 'Krishna Kanta Handiqui State Open University';

// outputs "KKHSOU, Krishna Kanta Handiqui State Open University"
echo "$my_variable, $my_Variable";
?>
```

In PHP, there is another way of assigning values to variables, "assign by reference". Here, the new variable simply references or points to the original variable. Therefore, changes to the new variable affects the original, and vice versa. To assign variables values by reference, simply prepend an ampersand "&" to the beginning of the source variable which is being assigned.

Example:

```
<?php
```

```
// Assigns the value 'KKHSOU' to $my_variable
$my_variable = 'KKHSOU';

// Reference $my_variable via $my_var
$my_var = &$my_variable;

// $my_var content is same as $my_variable
echo $my_var;

// $my_variable content is same as $my_var
echo $my_variable;

// Alter $my_var
$my_var = "I am from $my_var";

// $my_var is altered
echo $my_var;

// $my_variable is altered too
echo $my_variable;
?>
```

Though it is not necessary to initialize variables in PHP, it is a very good practice. Uninitialized variables have a default value of their type depending on the context in which they are used - booleans default to FALSE, integers and floats default to zero, strings are set as an empty string and arrays become to an empty array.

Scope of Variables

The scope of a variable is the context within which it is defined. Any variable used inside a function is by default limited to the local function scope.

Example:

```
<?php
$a = 1; //I am a global variable
```

```
$b = 2; //I am a global variable too

function Sum()
{
   global $a, $b; //declaring $a and $b global within the function
   $c = 4; //I am a local variable w.r.t. the Sum() method
   $b = $a + $b;
}

Sum(); //call to the function
echo $b; //I will get printed
echo '\n'; //insert a new line
echo $c; //I won't get printed
?>
```

Variable variables

A normal variable is set with a statement shown below.

```
<?php
$my_var = 'Hello';
?>
```

A *variable variable* takes the value of a variable and treats that as the name of a variable. In the above example, Hello, can be used as the name of a variable by using two dollar signs, as shown below.

```
<?php
$$my_var = 'KKHSOU';
?>
```

At this point two variables have been defined and stored in the PHP symbol tree: $my_var with the contents "Hello" and $Hello with the contents "KKHSOU".

Therefore, the statements shown below, produces the exact same output.

```
<?php
echo "$my_var ${$my_var}"; //prints “Hello KKHSOU”
```

```
echo "$my_var $Hello"; //prints “Hello KKHSOU”
?>
```

Variables from external sources

When a form is submitted to a PHP script, the information from that form is automatically made available to the script. There are many ways to access this information, one of the methods is shown below.

Let us take the example of the Login Form we had created in the earlier units.

```
<form action="process.php" method="post">
<table class="login-form">
<tr>
	<td>Message:</td>
	<td><input id="messageBox" type="text" name="messageBox" readonly></td>
</tr>
<tr>
	<td>Username:</td>
	<td><input id="username" type="text" name="username" onkeyup="checkUsername()"></td>
</tr>
<tr>
	<td>Password:</td>
	<td><input id="password" type="password" name="password"></td>
</tr>
<tr>
	<td colspan="2"><input id="login" type="submit" value="Login"></td>
</tr>
</table>
</form>
```

If you notice in the form above, on submit the form action is set to “**process.php**”. Let us create a PHP file named “process.php” and populate it with the content as shown below.

process.php

```
<?php
echo $_POST['messageBox'];
echo $_POST['username'];
echo $_POST['password'];
?>
```

Now, every time you submit the form, the values you had entered will be displayed to you by the “process.php”.

6.4.4 Constants

A constant is an identifier for a simple value. The value of a constant cannot change during the execution of the script. A constant is case-sensitive by default. By convention, constant identifiers are always uppercase. The name of a constant starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

Example:

```
<?php
define("UNIVERSITY", "KKHSOU");
?>
```

6.4.5 Operators

An operator is something that takes one or more values and yields another value. Operators can be grouped according to the number of values they take.

Unary operators take only one value. Examples of unary operators are the logical not operator "!", the increment operator "++", etc.

Binary operators take two values. Examples of binary operators are the plus "+", minus "-", etc.

Ternary operator takes three values. Example of a ternary operator is "?:".

Operator Precedence

The precedence of an operator specifies how "tightly" it binds two expressions together. For example, in the expression 1+5*3, the answer is 16 and not 18 because the multiplication "*" operator has a higher precedence than the addition "+" operator. Parentheses "()" may be used to force precedence, if necessary. For example, (1+5)*3 evaluates to 18.

When operators have equal precedence their associativity decides how the operators are grouped. For example, the minus "-" is *left-associative*, so 1-2-3 is grouped as (1-2)-3 and evaluates to -4. the equals-to "=" on the other hand is *right-associative*, so $a=$b=$c is grouped as $a=($b=$c). Furthermore, the “==” operator has lesser precedence than the “<=” operator.

No mater what the operator precedence, the use of parentheses “( )”, even when not strictly necessary, can often increase readability of the code by making grouping explicit rather than relying on the implicit operator precedence and associativity.

Arithmetic Operators

-$a : Negation, Opposite of $a.
$a + $b : Addition, Sum of $a and $b.
$a - $b : Subtraction, Difference of $a and $b.
$a * $b : Multiplication, Product of $a and $b.
$a / $b : Division, Quotient of $a and $b.
$a % $b : Modulus, Remainder of $a divided by $b.

Assignment Operators

The basic assignment operator is the "=" operator. The left operand gets set to the value of the expression on the right.

Example:

```
<?php
$a = 3; // $a gets set to the value 3
$a += 5; // sets $a to 8, same as $a = $a + 5;
$b = "Hello "; // $b gets set to the value "Hello"
$b .= "KKHSOU"; // sets $b to "Hello KKHSOU", same as $b = $b . "KKHSOU";
$c = &$a; // sets $c as a reference to $a
?>
```

Comparison Operators

The comparison operators allow you to compare two values.

$a == $b : Equal, TRUE if $a is equal to $b
$a === $b : Identical, TRUE if $a is equal to $b, and are of same data type
$a != $b : Not equal, TRUE if $a is not equal to $b
$a <> $b : Not equal, TRUE if $a is not equal to $b
$a !== $b : Not identical, TRUE if $a not equal to $b or not same data type
$a < $b : Less than, TRUE if $a is strictly less than $b
$a > $b : Greater than, TRUE if $a is strictly greater than $b
$a <= $b : Less than or equal to, TRUE if $a is less than or equal to $b
$a >= $b : Greater than or equal to, TRUE if $a greater than or equal to $b

Incrementing/Decrementing Operators

PHP supports C-style *pre*- and *post*- increment and decrement operators. The increment/decrement operators only affect numbers and strings. Arrays, objects and resources are not affected. Decrementing NULL values has no effect too, but incrementing NULL values results in 1.

++$a : Pre-increment, Increments $a by one, then returns $a
$a++ : Post-increment, Returns $a, then increments $a by one
--$a : Pre-decrement, Decrements $a by one, then returns $a
$a-- : Post-decrement, Returns $a, then decrements $a by one

Logical Operators

$a and $b : AND, TRUE if both $a and $b are TRUE
$a or $b : OR, TRUE if either $a or $b is TRUE
$a xor $b : XOR, TRUE if either $a or $b is TRUE, but not both
!$a : NOT, TRUE if $a is not TRUE
$a && $b : AND, TRUE if both $a and $b are TRUE
$a || $b : OR, TRUE if either $a or $b is TRUE

String Operators

There are two string operators. The first is the concatenation operator ".", which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ".=", which appends the argument on the right side to the argument on the left side.

Example:

```
<?php
$a = "Hello ";
$b = $a . "KKHSOU"; // now $b contains "Hello KKHSOU"
$a = "Hello ";
$a .= "KKHSOU";     // now $a contains "Hello KKHSOU"
?>
```

6.4.6 Control Structures

Any PHP script is built out of a series of statements. A statement can be an assignment, a function call, a loop, a conditional statement or even a statement that is an empty statement. Statements usually end with a semicolon ";". In addition, statements can be grouped into a statement-group by encapsulating a group of statements with curly braces "{ }". A statement-group is a statement by itself as well. We will just look at a few of them.

if, elseif, else

The **if** statement executes a block of code only if the specified condition is true.

Syntax for if statements:

if (*condition*) {

block of code

}

The **if, else** statement executes a block code if a condition is true and another block of code if the condition is false.

Syntax for if, else statements:

if (*condition*) {

block of code

} else {

another block of code

}

The **if, elseif, else** statement selects one of several blocks of code to be executed depending on the specified conditions.

Syntax for if, elseif, else statements:

if (*condition*) {

block of code

} elseif (*another condition*) {

some other block of code

} else {

another block of code

}

Example:

```
<?php
if ($a > $b) {
        echo "The value of a is greater than b";
} elseif ($a == $b) {
        echo "The value of a is equal to b";
} else {
        echo "The value of a is smaller than b";
}
```

```
?>
```

for

The PHP for loop executes a block of code a specified number of times. It contains the parameters "**init counter**" to initialize the loop counter value, the "**test counter**" to evaluated for each loop iteration (if this evaluates to TRUE, the loop continues else the loop ends) and the "**increment counter**" that increments the loop counter value.

Syntax:

for (init counter; test counter; increment counter)

{

code to be executed;

}

Example:

```
<?php
for ($i = 1; $i <= 10 ; $i++) {
	if ($i == 7) {
		echo "almost there... 'i=$i' now :-)<br />";
	}
	else {
		echo $i . '<br />';
	}
}
?>
```

include

The include statement includes and evaluates the specified file.

Example:

myvariables.php

```
<?php
$x = 'Welcome';
$y = 'KKHSOU';
?>
```

myphppage.php

```
<?php
echo "I said $x $y"; // prints only "I said" include 'myvariables.php';
echo "I said $x $y"; // prints out "I said Welcome KKHSOU"
?>
```

6.4.7 Functions

A function is a block of statements that can be used repeatedly in a program. It does not execute immediately when a page loads and will be executed only when called.

Example:

```
<?php
// function definition
function my_method()
{
        echo "Hello KKHSOU";
}
my_method(); // call the function my_method()
?>
```

Functions can be passed parameters as well, as show below.

```
<?php
// function definition
function my_method( $my_variable )
{
```

```
        echo "Hello $my_variable";
}
my_method( “KKHSOU” ); // call the function my_method()
?>
```

6.5 APPLYING WHAT WE HAVE LEARNT IX – CONNECTING TO MYSQL USING PHP

Before we can access the data stored in a database, we must open a connection to the MySQL server. We use the PHP **mysqli_connect()** function for this purpose. The syntax for this PHP function is shown below.

Syntax:

mysqli_connect(dbServer, username, password, dbName);

Here, the **dbServer** is the *hostname* or *IP Address* of the MySQL Server, **username** is the MySQL *username* to use for this connection, **password** is the MySQL *password* of the username used for this connection, **dbName** is the MySQL *database* to access in this connection.

We can use either of the methods, shown in the PHP code examples below, to connect to a MySQL database server.

Example:

```
<?php

//create a MySQL connection via UNIX socket
$mysqli = new mysqli('localhost', 'user', 'password', 'database');
```

```
//If there is a connection error display the error no. and error
if ($mysqli->connect_errno) {

        echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " .
$mysqli->connect_error;

}


//if the connection is successful display the connection host information
echo $mysqli->host_info;
 //prints “Localhost via UNIX socket”
mysqli_close($mysqli);

?>
```

Example:

```
<?php
//create a MySQL connection via TCP/IP PORT
$mysqli = new mysqli('127.0.0.1', 'user', 'password', 'database', 3306);


//If there is a connection error display the error no. and error
if ($mysqli->connect_errno) {

        echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " .
$mysqli->connect_error;

}


//if the connection is successful display the connection host information
echo $mysqli->host_info;
```

```
 //prints “127.0.0.1 via TCP/IP”
mysqli_close($mysqli);
?>
```

The connection will close when the script ends. However, it is good practice to close the connection using the PHP **mysqli_close()** function as shown in the examples above.

Just connecting to the MySQL Server will not serve our purpose, we will need to create a Database, a Table within the Database and a MySQL user with access to the database.

At the Linux/Windows Ternimal type in the following commands. In our examples, we are using the Fedora 18 distribution to perform all the commands shown. The typed in commands are marked as bold text.

To perform the database operations in our agenda, we need to login to MySQL Server as root.

```
[fedorauser@kkhsou ~]$ mysql -u root

Welcome to the MySQL monitor.  Commands end with ; or \g.

Your MySQL connection id is 51

Server version: 5.5.35 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
```

```
affiliates. Other names may be trademarks of their respective

owners.



Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.



mysql>
```

To see which databases are available, type in the command as shown below.

```
mysql> show databases;

+--------------------+

| Database           |

+--------------------+

| information_schema |

| mysql              |

| performance_schema |
```

```
| test               |

+--------------------+

4 rows in set (0.00 sec)

mysql>
```

These are the default databases available after the MySQL installation.

To create a database named "kkhsou", type in the command as shown below.

```
mysql> CREATE DATABASE kkhsou;

Query OK, 1 row affected (0.00 sec)

mysql>
```

To create a user named "kkhsou" with the password "12345678", type in the command as shown below.

```
mysql> CREATE USER 'kkhsou'@'localhost' IDENTIFIED BY '12345678';

Query OK, 0 rows affected (0.00 sec)

mysql>
```

To grant access of the database "kkhsou" to the MySQL user "kkhsou" with the password "12345678", type in the command as shown below.

```
mysql> GRANT ALL ON kkhsou.* TO 'kkhsou' IDENTIFIED BY '12345678';

Query OK, 0 rows affected (0.00 sec)

mysql>
```

To see which databases are available now, type in the command again as shown below.

```
mysql> show databases;

+--------------------+

| Database           |

+--------------------+

| information_schema |

| kkhsou             |

| mysql              |
```

```
| performance_schema |

| test               |

+--------------------+

5 rows in set (0.00 sec)

mysql>
```

To use the database named "kkhsou", type in the command as shown below.

```
mysql> use kkhsou;

Database changed

mysql>
```

To see the tables within the database "kkhsou", type in the command as shown below.

```
mysql> show tables;

Empty set (0.00 sec)
```

```
mysql>
```

To create a table within the database "kkhsou", type in the command as shown below.

```
mysql> CREATE TABLE books( title VARCHAR(30), author VARCHAR(30), publisher VARCHAR(30), year VARCHAR(30) );

Query OK, 0 rows affected (0.08 sec)



mysql>
```

To see the tables within the database "kkhsou" available now, type in the command as shown below.

```
mysql> show tables;

+------------------+

| Tables_in_kkhsou |

+------------------+

| books            |
```

```
+-----------------+

1 row in set (0.00 sec)



mysql>
```

To view the contents of the table "books", type in the command as shown below.

```
mysql> select * from books;

Empty set (0.00 sec)



mysql>
```

To insert values into the table "books", type in the commands one-by-one as shown below.

```
mysql> INSERT INTO books VALUES ("Title A", "Author A", "Publisher A", "2001");

Query OK, 1 row affected (0.02 sec)
```

mysql> **INSERT INTO books VALUES ("Title B", "Author B", "Publisher B", "2002");**

Query OK, 1 row affected (0.04 sec)

mysql> **INSERT INTO books VALUES ("Title C", "Author C", "Publisher C", "2003");**

Query OK, 1 row affected (0.02 sec)

mysql> **INSERT INTO books VALUES ("Title D", "Author D", "Publisher D", "2004");**

Query OK, 1 row affected (0.03 sec)

mysql> **INSERT INTO books VALUES ("Title E", "Author E", "Publisher E", "2005");**

Query OK, 1 row affected (0.02 sec)

mysql> **INSERT INTO books VALUES ("Title F", "Author F", "Publisher F", "2006");**

Query OK, 1 row affected (0.03 sec)

```
mysql> INSERT INTO books VALUES ("Title G", "Author G", "Publisher G",
"2007");

Query OK, 1 row affected (0.02 sec)



mysql>
```

To view the contents of the table "books" available now, type in the command as shown below.

```
mysql> select * from books;

+---------+----------+-------------+------+

| title   | author   | publisher   | year |

+---------+----------+-------------+------+

| Title A | Author A | Publisher A | 2001 |

| Title B | Author B | Publisher B | 2002 |

| Title C | Author C | Publisher C | 2003 |

| Title D | Author D | Publisher D | 2004 |
```

```
| Title E | Author E | Publisher E | 2005 |

| Title F | Author F | Publisher F | 2006 |

| Title G | Author G | Publisher G | 2007 |

+---------+----------+-------------+------+

7 rows in set (0.00 sec)



mysql>
```

6.6 APPLYING WHAT WE HAVE LEARNT X – BUILDING A WEB PAGE USING PHP

In section **6.4.3 Variables**, we had used an example to understand the "**Variables from external sources**" and created a "**process.php**" file. The HTML form when submitted, passed on its form values to the "process.php" script. In this section, we will work with this file and add the following PHP code, as shown below.

```
<?php

//get the HTML form values

$html_messageBox = $_POST['messageBox']; //global variable

$html_username = $_POST['username']; //global variable
```

```
$html_password = $_POST['password']; //global variable



if( $html_username === '' ) {

       $html_username = '--- EMPTY ---';

}



if( $html_password === '' ) {

        $html_password = '--- EMPTY ---';

}



echo 'Hello there! you have submitted the following to me.<br />'; //using ''

echo "Username = <strong>$html_username</strong><br />"; //using ""

echo "Password = <strong>$html_password</strong><br />"; //using ""

echo 'To try again click <a href="index.html">here</a>.<br />'; //using ''







if( $html_username === 'kkhsou' && $html_password === 'abcd1234' ) {
```

```
	echo '<h3>This is a restricted content</h3>';



	//create a MySQL connection via TCP/IP PORT

	$mysqli = new mysqli('127.0.0.1', 'kkhsou', '12345678', 'kkhsou', 3306);



	//If there is a connection error display the error no. and error

	if ($mysqli->connect_errno) {



		echo "Failed to connect to MySQL: (" . $mysqli->connect_errno .
") " . $mysqli->connect_error;

	}



	//if the connection is successful display the connection host
information

	echo 'Connected to MySQL ' . $mysqli->host_info . '<br />'; //prints
"127.0.0.1 via TCP/IP"
```

```
/* we are connected to the MySQL Server now

  and therefore can perform some Database

  related tasks */



$result = mysqli_query($mysqli,"SELECT * FROM books;");



echo 'The table below is from a MySQL Database<br />';



echo '<table border="1" cellpadding="5" cellspacing="0">';

echo '<tr>';

echo '<th>Title</th><th>Author</th><th>Publisher</th><th>Year</th>';

echo '</tr>';



while($tablerow = mysqli_fetch_array($result))

{

       echo "<tr>";
```

```
            echo "<td>" . $tablerow['title'] . "</td>";

            echo "<td>" . $tablerow['author'] . "</td>";

            echo "<td>" . $tablerow['publisher'] . "</td>";

            echo "<td>" . $tablerow['year'] . "</td>";

            echo "</tr>";

        }



        echo '</table>';



        /* we are done with our database related work

           so the database connection can be closed */



        //close the MySQL connection

        mysqli_close($mysqli);

        echo 'Closed the connection to MySQL';



} elseif( $html_username === 'fedora' && $html_password === 'fedora' ) {
```

```
       echo '<h3>This content is for Fedora User</h3>';

       echo '<blockquote cite="https://fedoraproject.org/en/about-fedora"><p
align="justify">Fedora is a Linux-based operating system, a collection of
software that makes your computer run. You can use Fedora in addition to, or
instead of, other operating systems such as Microsoft Windows™ or Mac OS X™.
The Fedora operating system is completely free of cost for you to enjoy and
share.</p><p align="justify">The Fedora Project is the name of a worldwide
community of people who love, use, and build free software. We want to lead in
the creation and spread of free code and content by working together as a
community. Fedora is sponsored by Red Hat, the world\'s most trusted provider
of open source technology. Red Hat invests in Fedora to encourage collaboration
and incubate innovative new free software technologies.</p><p
align="justify">Please click <a href="https://fedoraproject.org/en/about-
fedora" target="_anotherTab">here</a> for more information.</p></blockquote>';



} elseif( $html_username === 'fsf' && $html_password === 'fsf' ) {



        echo '<h3>This content is for FSF User</h3>';

       echo '<blockquote cite="http://www.fsf.org/about/"><p
align="justify">The free software movement is one of the most successful social
movements to emerge in the past 25 years, driven by a worldwide community of
ethical programmers dedicated to the cause of freedom and sharing. But the
ultimate success of the free software movement depends upon teaching our
friends, neighbors and work colleagues about the danger of not having software
```

```
freedom, about the danger of a society losing control over its computing.</p><p
align="justify">Please click <a href="http://www.fsf.org"
target="_anotherTab">here</a> for more information.</p></blockquote>';



} else {



       echo '<h3>Maybe you shouldn\'t be here :-)</h3>';



}



?>
```

Once, you have populated the “process.php” file with the contents shown above, try the below mentioned username/password combinations in the Login Form by opening the link “http://localhost/index.html”.

Username: kkhsou

Password: abcd1234

Username: fedora

Password: fedora

Username: fsf

Password: fsf

If you have followed properly all the examples shown in this unit, you will get different content based on the username/password combinations entered and submitted from the HTML form.

6.7 LET US SUM UP

In this unit, we have briefly discussed the basics of PHP. Though we have covered only a few of both the topics, these topics are intended to inspire and point you in a direction to further explore. This unit was not intended to provide you with an exhaustive in-depth on the topics but merely to introduce you to some of the basic concepts of PHP and use them in web pages.

What we have learned in this unit.

- PHP basics
- Connecting to MySQL using PHP
- Using PHP in a Web page

CHECK YOUR PROGRESS

Q1: PHP is

(a) a web page (b) a scripting language

(c) a css file (d) a markup language

Q2: Which of the following file extension(s) is(are) used for a PHP file?

(a) .html (b) .js

(c) .xml (d) .php

Q3: Which of the following is a correct syntax for comments in PHP?
(a) <!-- this is a HTML comment --> (b) /* this is a XML comment */
(c) // this is a JavaScript comment (d) # this is a PHP comment

Q4: For PHP code, which of the following processing tags are used?
(a) “<script> … </script>” tags (b) ”<?php … ?>” tags
(c) “<noscript> … </noscript>” tags (d) “<xml></xml>” tags

Q5: Which of the following location contains the default **DocumentRoot** for the Apache Web Server?
(a) /var/www/html (b) /var/log/httpd
(c) /etc/httpd (d) /etc/www/html

Q6: What is the purpose of the command **su -c "yum install mysql-server"**?

Q7: Which of the following data types are compound data types?
(a) boolean (b) string
(c) object (d) array

Q8: How can you assign variables by reference in PHP?

Q9: How can you explicitly override operator precedence in PHP?

Q10: What does the PHP expression **$a === $b** mean?

6.9 FURTHER READINGS

- W. Jason Gilmore, "Beginning PHP and MySQL: From Novice to Professional 4th Edition", Apress, ISBN-13 9788184897456, ISBN-10 8184897456, 2010

- Steven Holzner, "PHP: The Complete Reference 1st Edition", Tata Mcgraw Hill Education Private Limited, ISBN-13 9780070223622, ISBN-

10 0070223629, 2007

- http://www.php.net/manual/en/
- http://en.wikipedia.org/wiki/PHP
- http://in2.php.net/FAQ.php

6.8 ANSWERS TO CHECK YOUR PROGRESS

A1: (b)
A2: (d)
A3: (b), (c), (d)
A4: (b)
A5: (a)
A6: This command installs the server and client components of the MySQL Server.
A7: (c), (d)
A8: Assigning variables by reference is done by using the & operator. For example, **$my_var = &$my_variable**
A9: We can use set of parentheses "( )" to override the implicit operator precedence in PHP.
A10: Identical, TRUE if $a is equal to $b, and are of the same data type as well.

6.10 MODEL QUESTIONS

1. What is PHP and what is it used for?
2. Which are the start and end tags that the PHP parser reads to start parsing and stop parsing PHP code?
3. How do we test if the **php-mysql** module is available and working?

4. To specify a boolean literal which case-insensitive keywords are used?
5. What is an array?
6. What does scope of a variable mean in PHP??
7. In the heredoc syntax, the identifier must follow the same naming rules as a label. What are these?
8. How can integers be specified in octal notation?
9. How can the methods of a class be used in PHP?
10. Which symbol/sign represents the start of a variable name in PHP?

****

UNIT 7: CREATING A WEB APPLICATION – PUTTING IT ALL TOGETHER

UNIT STRUCTURE

7.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about the MVC framework
- learn a few coding considerations
- create a Web Application using the MVC framework

7.2 INTRODUCTION

In the early days of PHP web applications, fragments of PHP code were mixed in with HTML mark-up. There were no frameworks, so Web applications were just a bunch of source files. As the PHP language matured, web applicaton

developers started to think about the cleanliness and maintainability of their code. The model-view-controller (MVC) pattern was therefore introduced. MVC is a software architecture that allows for the separation of business logic from the user interface.

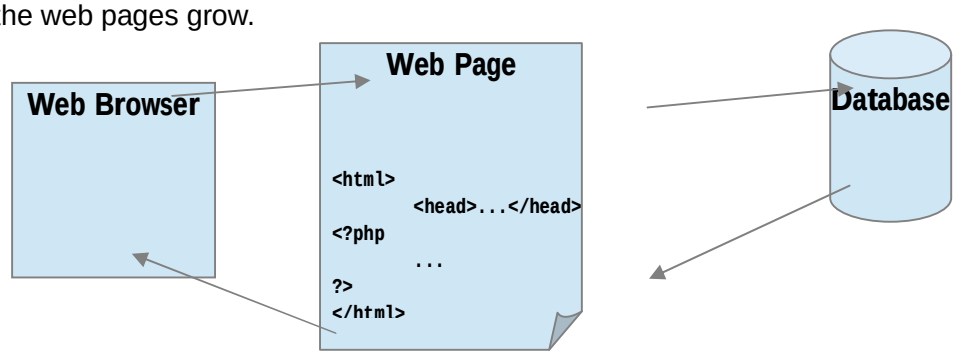
In this unit, we will briefly look at the MVC framework and use its design pattern to build our Web Application.

7.3 THE MVC DESIGN PATTERN

In the MVC architecture, the user sees and interacts with the **view** that is generated by HTML code along with JavaScript, CSS, images, etc.
The user actions in the web browser are passed as HTTP requests (GET or POST methods) to the **controller**. The controller is a piece of code that handles and processes the user inputs and then reads and makes necessary changes to the **model** which is responsible for the storage and modification of data. In simple terms, the model consists of the database structure and contents, and the code used to access it. Thereafter, the controller generates the proper view that will be sent and displayed to the user agent.

7.3.1 Basic Web Architecture

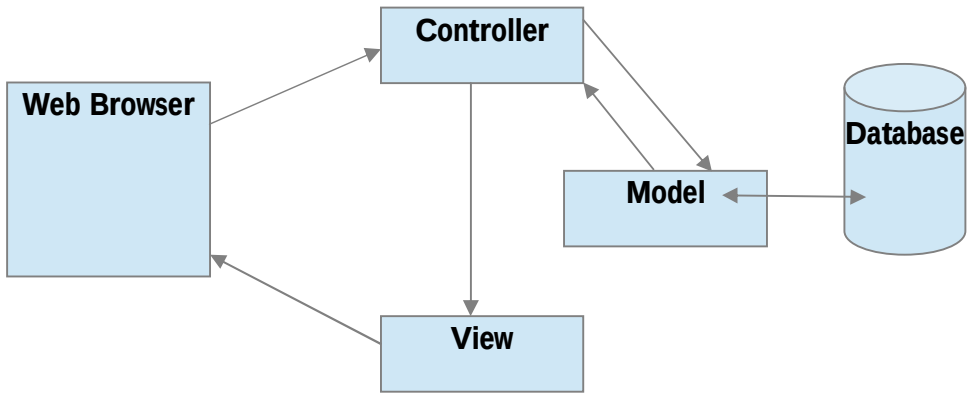
In the basic web architecture, as depicted in the figure below, the user agent or the web browser interacts with a web page that contains within it code for checking user inputs, database query, displaying the results, etc. Though this architecture works perfectly well, it becomes difficult for maintaining the code as the web pages grow.



With the growing web applications on the Web it became imperative for web application developers to seek refuge in application development frameworks that had the functional abstraction for ease of code readability and maintainability. Among the many framework architectures available for web application development, we will cover in this unit the MVC framework that is widely accepted and used for web application development.

7.3.2 MVC Architecture

In the MVC architecture, as depicted in the figure below, all user agent or browser requests are handled by the **controller** which takes decisions for handling the requests. The controller then interacts with the **model**, which handles data and interacts with the database. The controller also interacts with the **view**, which handles the presentation, to fulfill the browser requests.



Therefore, all the user inputs are handled by the controller, the model does the processing and the output is handled by the view.

In the next section of this unit, we will briefly touch upon some points that we should consider while writing our code for improving code readability, code maintainability.

7.3.3 Coding Considerations

While writing code for your programs, you should consider the following points.

Name your variables properly: If you give your variables descriptive names, it will not require the extra comments in your code. Though you should always avoid absurdly long names which will can defeat the purpose in the first place. The same would also apply while naming functions, methods, classes, etc. too.

Comments should be used sparingly: Though comments are an important part in the whole "*code understandability / readability*" thing, you should never overdo it. Think about the developers who will be reading your code, it should not be a daunting task for them.

Code Style and consistency: The style in which you prefer to write your code should be consistent though out your applications. This makes the code both understandable and readable by the other developers who may be looking at your code.

Adhering to the above listed coding considerations are not mandatory for writing your code. However, this is the era of collaborative application development and making your code readable and style consistent is the need of the hour.

In all the previous units covered thus far, we have used code examples and covered some software prerequisites that we will use to build our MVC framework. Therefore, before we proceed further here is a quick check list of the software prerequisites assumed to be installed or available to us. The installation of these have already been covered in the previous unit. We will also deal with some of the additional configurations that will be required and discuss where ever necessary.

7.4 SETTING UP OUR DEVELOPMENT ENVIRONMENT

The following should be installed and available for use on Fedora 18 to proceed.

- Apache Web Server v2.4.x
- PHP/5.4.X
- MySQL 5.5.X

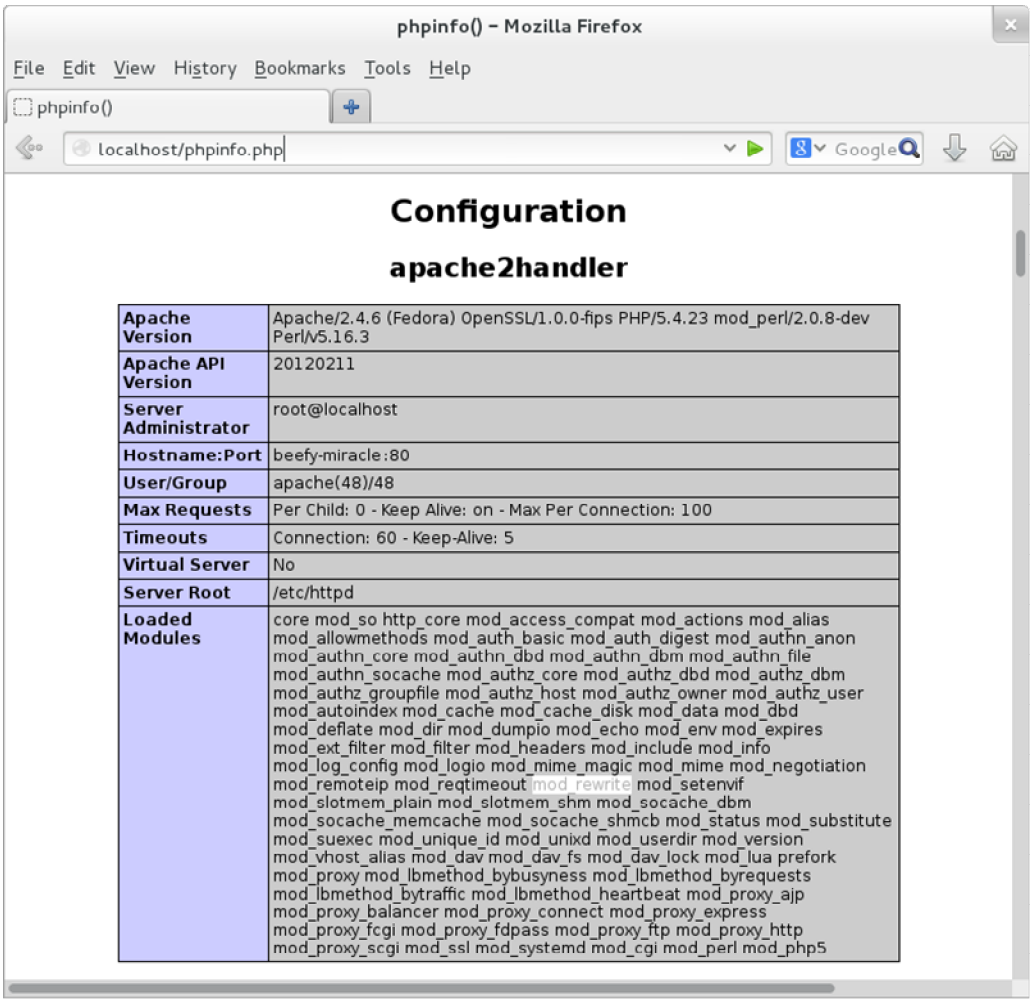
Some, additional steps that are needed to be performed are listed as below. The steps listed below are for the Fedora 18 distribution of Linux.

- Confirm that the Apache **mod_rewrite** module is loaded and available.

The **/etc/httpd/conf.modules.d/00-base.conf** file should contain the following line:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

The phpinfo() should display the mod_rewrite module as shown below:



- We will be using the ".htaccess" file for defining our URL rewrite rules and therefore the web folder containing our PHP code will require the following configuration to be in place.

The "**<Directory "/var/www/html">**" section within the main Apache Server configuration file **/etc/httpd/conf/httpd.conf** should look something like this as shown below, with the

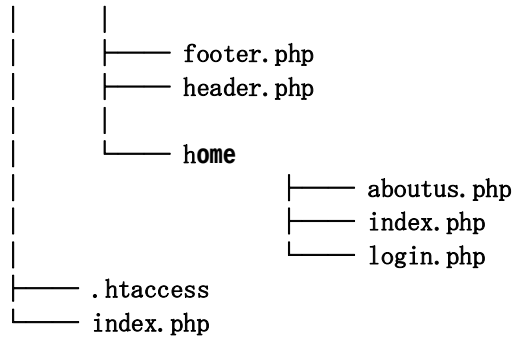
"**`AllowOverride All`**" set for the "/var/www/html" folder. As this folder will contain our MVC web application.

```
# Further relax access to the default document root:
<Directory "/var/www/html">
    #
    # Possible values for the Options directive are "None", "All",
    # or any combination of:
    #   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
```

```
    #
    # Note that "MultiViews" must be named *explicitly* --- "Options All"
    # doesn't give it to you.
    #
    # The Options directive is both complicated and important.  Please see
    # http://httpd.apache.org/docs/2.4/mod/core.html#options
    # for more information.
    #
    Options Indexes FollowSymLinks
    #
    # AllowOverride controls what directives may be placed in .htaccess files.
    # It can be "All", "None", or any combination of the keywords:
    #   Options FileInfo AuthConfig Limit
    #
    AllowOverride All
    #
    # Controls who can get stuff from this server.
    #
    Require all granted
</Directory>
```

- Our code will reside inside the folder "**/var/www/html/mymvc**", and we will use the following directory structure to store our code files in.

```
/var/www/html/mymvc/
|
├─── application
|       |
|       ├─── config
|       |       └─── config.php
|       |
|       ├─── controller
|       |       ├─── books.php
|       |       └─── home.php
|       |
|       ├─── lib
|       |       ├─── application.php
|       |       └─── controller.php
|       |
|       ├─── model
|       |       └─── booksmodel.php
|       |
|       └─── view
|       ├─── books
|       |       └─── index.php
```



7.5 BUILDING OUR MVC FRAMEWORK

To start building our MVC framework, we will navigate to the "/var/www/html" folder. This is the default **DocumentRoot** for Apache and also it requires root privileges. Therefore, we will use the "su -" command first and type in the root password in the terminal window, as shown below.

```
[fedorauser@kkhsou ~]$ su -
Password:
[root@kkhsou ~]#
[root@kkhsou ~]# cd /var/www/html/
[root@kkhsou html]#
```

Next, we need to create a folder within the **DocumentRoot** to contain our code. Let us name this folder as "mymvc".

```
[root@kkhsou html]# mkdir mymvc
[root@kkhsou html]# cd mymvc
[root@kkhsou mymvc]#
```

We will create the directory structure, as shown in the previous section, to store our code files as we go along likewise.

CREATING THE .htaccess FILE

Create a "**.htaccess**" file within the "**/var/www/html/mymvc/**" folder, with the contents as shown below. The purpose of this file is to enable a single point of entry into our mvc framework.

.htaccess

```
# Prevents issues with controller named "index" and having a root index.php
# more here: http://httpd.apache.org/docs/2.2/content-negotiation.html
Options -MultiViews

# Activates URL rewriting (like kkhsou.com/controller/action/1/2/3)
RewriteEngine On

# Disallows others to look directly into /public/ folder
Options -Indexes

# When using the script within a sub-folder, put this path here
# If your app is in the root of your web folder, then leave it commented out
RewriteBase /mymvc/

# General rewrite rules
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule ^(.+)$ index.php?url=$1 [QSA,L]
```

CREATING THE application FOLDER

Now, create a folder named "**application**" within "**/var/www/html/mymvc/**". Then, within this "application" folder create a "**config**" folder and add the file "config.php" with the contents as shown below.

```
<?php
/*
 * Configuration
 *
 * Please refer to http://php.net/manual/en/function.define.php
 */
/*
 * Configuration for: Error reporting
 * Useful to show problems during development.
 */
error_reporting(E_ALL);
ini_set("display_errors", 1);

/*
 * Configuration for: Project URL
 * Put your URL here
 */
define('URL', 'http://localhost/mymvc/');

/*
 * Configuration for: Database
 * Define your database credentials, database type etc.
 */
define('DB_TYPE', 'mysql');
```

```
define('DB_HOST', '127.0.0.1');
define('DB_NAME', 'mymvc');
define('DB_USER', 'root');
define('DB_PASS', 'yourpassword');
?>
```

CREATING THE DATABASE

Create a MySQL database named "**mymvc**" and add the following tables to it, as shown below.

```
[root@kkhsou mymvc]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.35 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
mysql> CREATE DATABASE IF NOT EXISTS `mymvc`;
mysql>
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| kkhsou             |
| mymvc              |
| mysql              |
| performance_schema |
| test               |
+--------------------+
6 rows in set (0.00 sec)

mysql>
mysql> use mymvc
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
mysql> CREATE TABLE `mymvc`.`books` (
    ->   `id` int(11) NOT NULL AUTO_INCREMENT,
```

```
    ->   `title` text COLLATE utf8_unicode_ci NOT NULL,
    ->   `author` text COLLATE utf8_unicode_ci NOT NULL,
    ->   `publisher` text COLLATE utf8_unicode_ci NOT NULL,
    ->   `year` text COLLATE utf8_unicode_ci NOT NULL,
    ->   PRIMARY KEY (`id`),
    ->   UNIQUE KEY `id` (`id`)
    ->    )    ENGINE=InnoDB    AUTO_INCREMENT=31    DEFAULT    CHARSET=utf8
COLLATE=utf8_unicode_ci;
Query OK, 0 rows affected (0.10 sec)

mysql>
mysql> INSERT INTO `mymvc`.`books` (`id`, `title`, `author`, `publisher`,
`year`) VALUES
    -> (1, 'Title A', 'Author A', 'Publisher A', '2001'),
    -> (2, 'Title B', 'Author B', 'Publisher B', '2002'),
    -> (3, 'Title C', 'Author C', 'Publisher C', '2003'),
    -> (4, 'Title D', 'Author D', 'Publisher D', '2004'),
    -> (5, 'Title E', 'Author E', 'Publisher E', '2005'),
    -> (6, 'Title F', 'Author F', 'Publisher F', '2006'),
    -> (7, 'Title G', 'Author G', 'Publisher G', '2007'),
    -> (8, 'Title H', 'Author H', 'Publisher H', '2008'),
    -> (9, 'Title I', 'Author I', 'Publisher I', '2009'),
    -> (10, 'Title J', 'Author J', 'Publisher J', '2010'),
    -> (11, 'Title K', 'Author K', 'Publisher K', '2011'),
    -> (12, 'Title L', 'Author L', 'Publisher L', '2012');
Query OK, 12 rows affected (0.04 sec)
Records: 12  Duplicates: 0  Warnings: 0

mysql>
mysql> select * from books;
+----+---------+----------+-------------+------+
| id | title   | author   | publisher   | year |
+----+---------+----------+-------------+------+
|  1 | Title A | Author A | Publisher A | 2001 |
|  2 | Title B | Author B | Publisher B | 2002 |
|  3 | Title C | Author C | Publisher C | 2003 |
|  4 | Title D | Author D | Publisher D | 2004 |
|  5 | Title E | Author E | Publisher E | 2005 |
|  6 | Title F | Author F | Publisher F | 2006 |
|  7 | Title G | Author G | Publisher G | 2007 |
|  8 | Title H | Author H | Publisher H | 2008 |
|  9 | Title I | Author I | Publisher I | 2009 |
| 10 | Title J | Author J | Publisher J | 2010 |
| 11 | Title K | Author K | Publisher K | 2011 |
| 12 | Title L | Author L | Publisher L | 2012 |
+----+---------+----------+-------------+------+
12 rows in set (0.00 sec)

mysql>
```

```
mysql> CREATE TABLE `mymvc`.`users` (
    ->   `id` int(11) NOT NULL AUTO_INCREMENT,
    ->   `username` varchar(32) NOT NULL,
    ->   `password` varchar(32) NOT NULL,
    ->   PRIMARY KEY (`id`),
    ->   UNIQUE KEY `id` (`id`)
    ->    )    ENGINE=InnoDB    AUTO_INCREMENT=31    DEFAULT    CHARSET=utf8
COLLATE=utf8_unicode_ci;
Query OK, 0 rows affected (0.07 sec)

mysql>
mysql> INSERT INTO `mymvc`.`users` (`id`, `username`, `password`) VALUES
    -> (1, 'kkhsou', md5('"abcd1234" . "@kKh|+-|SoU%"'));
Query OK, 1 row affected (0.05 sec)

mysql>
mysql> select * from users;
+----+----------+----------------------------------+
| id | username | password                         |
+----+----------+----------------------------------+
|  1 | kkhsou   | 76133bf68dd9d2bbba9f36e4498c92bf |
+----+----------+----------------------------------+
1 row in set (0.00 sec)

mysql>
```

Alternatively, you could also upload the following SQL file from the MySQL prompt, by typing the command "**`mysql -u root -p < database-tables.sql`**".

database-tables.sql

```
CREATE DATABASE IF NOT EXISTS `mymvc`;

CREATE TABLE `mymvc`.`books` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` text COLLATE utf8_unicode_ci NOT NULL,
  `author` text COLLATE utf8_unicode_ci NOT NULL,
  `publisher` text COLLATE utf8_unicode_ci NOT NULL,
  `year` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=31 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

CREATE TABLE `mymvc`.`users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(32) NOT NULL,
  `password` varchar(32) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=31 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

```
INSERT INTO `mymvc`.`books` (`id`, `title`, `author`, `publisher`, `year`)
VALUES
(1, 'Title A', 'Author A', 'Publisher A', '2001'),
(2, 'Title B', 'Author B', 'Publisher B', '2002'),
(3, 'Title C', 'Author C', 'Publisher C', '2003'),
(4, 'Title D', 'Author D', 'Publisher D', '2004'),
(5, 'Title E', 'Author E', 'Publisher E', '2005'),
(6, 'Title F', 'Author F', 'Publisher F', '2006'),
(7, 'Title G', 'Author G', 'Publisher G', '2007'),
(8, 'Title H', 'Author H', 'Publisher H', '2008'),
(9, 'Title I', 'Author I', 'Publisher I', '2009'),
(10, 'Title J', 'Author J', 'Publisher J', '2010'),
(11, 'Title K', 'Author K', 'Publisher K', '2011'),
(12, 'Title L', 'Author L', 'Publisher L', '2012');

INSERT INTO `mymvc`.`users` (`id`, `username`, `password`) VALUES (1, 'kkhsou',
md5('"abcd1234" . "@kKh|+-|SoU%"'));
```

CREATING THE index.php FILE

Next, we create an "**index.php**" file with the contents shown below, inside the folder "**/var/www/html/mymvc/**". This file will load all our configurations and classes.

index.php

```
<?php
/*
 * A simple MVC skeleton
 */

// load application config (error reporting etc.)
require 'application/config/config.php';

// load application class
require 'application/lib/application.php';
require 'application/lib/controller.php';

// start the application
$app = new Application();

?>
```

CREATING THE DEFAULT CLASSES

Create the application and controller classes and put them in the "**/var/www/html/mymvc/application/lib/**" folder. For this we create two separate files "**application.php**" and "**controller.php**", and populate them with the

respective contents shown below.

../lib/application.php

```
<?php

class Application
{
    /* The controller set to null */
    private $url_controller = null;

    /* The method (of the above controller) set to null */
    private $url_action = null;

    /* Set the Parameter one to null */
    private $url_parameter_1 = null;

    /* Set the Parameter two to null */
    private $url_parameter_2 = null;

    /* Set the Parameter three to null */
    private $url_parameter_3 = null;

    /*
     * "Start" the application:
     * Analyzes the URL elements and calls accordingly
     * the controller/method or the fallback
     */

    public function __construct()
    {
        // create array with URL parts in $url
        $this->splitUrl();

        // check controller: does such a controller exist ?
        if (file_exists('./application/controller/' . $this->url_controller . '.php')) {

            // if so, then load this file and create this controller
            // example: if controller would be "login", then this line
             // would translate into: $this->login = new login();
            require './application/controller/' . $this->url_controller . '.php';
            $this->url_controller = new $this->url_controller();

            // check method: does such a method exist in the controller ?
            if (method_exists($this->url_controller, $this->url_action)) {


                // call the method and pass the arguments to it
                if (isset($this->url_parameter_3)) {
                          //  will  translate  to  something  like  $this->home->method($param_1,
$param_2, $param_3);
                          $this->url_controller->{$this->url_action}($this->url_parameter_1,
$this->url_parameter_2, $this->url_parameter_3);

                } elseif (isset($this->url_parameter_2)) {

                          //  will  translate  to  something  like  $this->home->method($param_1,
$param_2);
                          $this->url_controller->{$this->url_action}($this->url_parameter_1,
$this->url_parameter_2);

                } elseif (isset($this->url_parameter_1)) {
```

```
                        // will translate to something like $this->home->method($param_1);
                        $this->url_controller->{$this->url_action}($this->url_parameter_1);

                } else {
                        // if no parameters given, just call the method without parameters, like
$this->home->method();
                        $this->url_controller->{$this->url_action}();


                }

            } else {

                 // default/fallback: call the index() method of a selected controller
                 $this->url_controller->index();

            }

        } else {

            // invalid URL, so simply show home/index
            require './application/controller/home.php';


            $home = new Home();


            $home->index();
        }
    }

    /*
     * Get and split the URL
     */
    private function splitUrl()
    {
        if (isset($_GET['url'])) {

            // split URL
            $url = rtrim($_GET['url'], '/');

            $url = filter_var($url, FILTER_SANITIZE_URL);

            $url = explode('/', $url);

            // Put URL parts into according properties
            //  By  the  way,  the  syntax  here  is  just  a  short  form  of  if/else,  called  "Ternary
Operators"
            $this->url_controller = (isset($url[0]) ? $url[0] : null);
            $this->url_action = (isset($url[1]) ? $url[1] : null);
            $this->url_parameter_1 = (isset($url[2]) ? $url[2] : null);
            $this->url_parameter_2 = (isset($url[3]) ? $url[3] : null);
            $this->url_parameter_3 = (isset($url[4]) ? $url[4] : null);

            // for debugging. uncomment this if you have problems with the URL
            // echo 'Controller: ' . $this->url_controller . '<br />';
            // echo 'Action: ' . $this->url_action . '<br />';
            // echo 'Parameter 1: ' . $this->url_parameter_1 . '<br />';
            // echo 'Parameter 2: ' . $this->url_parameter_2 . '<br />';
            // echo 'Parameter 3: ' . $this->url_parameter_3 . '<br />';
        }

    }
```

```
}
```

../lib/controller.php

```
<?php
/*
 * This is the "base controller class".
 * All other "real" controllers extend this class.
 */

class Controller
{
    /*
     * The Database Connection is set to null
     */
    public $db = null;
    /*
     * Whenever a controller is created, open
     * a database connection too. The idea behind
     * this is to have ONE connection that can be
     * used by multiple models (there are frameworks
     * that open one connection per model).
     */
    function __construct()
    {
        $this->openDatabaseConnection();
    }

    /*
     * Open the database connection with the credentials
     * from the application/config/config.php file
     */
    private function openDatabaseConnection()
    {
        // set the (optional) options of the PDO connection.
        // In this case, we set the fetch mode to "objects",
        // which means all results will be objects, like this:
        // $result->username
        // For example, the fetch mode FETCH_ASSOC would return
        // results like this: $result["username"]
        // see http://www.php.net/manual/en/pdostatement.fetch.php


        $options  =  array(PDO::ATTR_DEFAULT_FETCH_MODE  =>  PDO::FETCH_OBJ,  PDO::ATTR_ERRMODE  =>
PDO::ERRMODE_WARNING);

        // generate a database connection and using the PDO connector
        $this->db = new PDO(DB_TYPE . ':host=' . DB_HOST . ';dbname=' . DB_NAME, DB_USER, DB_PASS,
$options);

    }

    /*
     * Load the model with the given name.
     * loadModel("BooksModel") would include models/booksmodel.php
     * and create the object in the controller, like this:
     * $books_model = $this->loadModel('BooksModel');
     * Note that the model class name is written in "CamelCase",
     * the model's filename is the same in lowercase letters
     * param string $model_name The name of the model
     * return object model
```

```
     */

    public function loadModel($model_name)
    {
        require 'application/model/' . strtolower($model_name) . '.php';

        // return new model (and pass the database connection to the model)
        return new $model_name($this->db);

    }

}
```

CREATING AND ADDING SOME CONTROLLERS

We have not yet created a controller and therefore let us go ahead and create a default controller called “home”, which will also be our default application page. We will create the file named “**home.php**” for this controller and populate it with the content as shown below. This file is saved inside the folder “**/var/www/html/mymvc/application/controller/**”.

../controller/home.php

```
<?php



/*

 * Class Home

 *

 * Please note:

 * Don't use the same name for class and method,

 * as this might trigger an (unintended) __construct

 * of the class. This is really weird behaviour, but

 * documented here:

 * http://php.net/manual/en/language.oop5.decon.php

 *

 */

class Home extends Controller

{

    /*

     * PAGE: index

     * This method handles what happens when you move
```

```
     * to http://yourproject/home/index (which is the
     * default page)
     */
    public function index()
    {
        // debug message to show where you are, just for the demo
        echo 'Controller says: You are in the controller home, using the method index()';
        // load some views
        require 'application/view/header.php';
        require 'application/view/home/index.php';
        require 'application/view/footer.php';
    }


    /*
     * PAGE: login
     * This method handles what happens when you move to
     * http://localhost/home/login
     * The camelCase writing is just for better readability.
     * The method name is case insensitive.
     */
    public function logIn()
    {
        // debug message to show where you are, just for the demo
        echo 'Controller says: You are in the controller home, using the method logIn()';
        // load some views
        require 'application/view/header.php';
        require 'application/view/home/login.php';
        require 'application/view/footer.php';
    }
```

```
    /*
     * PAGE: aboutus
     * This method handles what happens when you move to
     * http://localhost/home/aboutus
     * The camelCase writing is just for better readability.
     * The method name is case insensitive.
     */
    public function aboutUs()
    {
        // debug message to show where you are, just for the demo
        echo 'Controller says: You are in the controller home, using the method aboutUs()';
        // load some views
        require 'application/view/header.php';
        require 'application/view/home/aboutus.php';
        require 'application/view/footer.php';
    }

}

?>
```

Let us create another controller called “**books**” as well, as shown below.

../controller/books.php

```
<?php

/*
 * Class Books
 *
 * Please note:
 * Don't use the same name for class and method,
 * as this might trigger an (unintended) __construct
```

```
 * of the class.
 * This is really weird behaviour, but documented here:
 * http://php.net/manual/en/language.oop5.decon.php
 *
 */
class Books extends Controller
{
    /*
     * PAGE: index
     * This method handles what happens when you move to
     * http://localhost/mymvc/books/index
     */
    public function index()
    {
        // simple message to show where you are
        echo 'Controller says: You are in the Controller <strong>Books</strong>, using the method
index().';

        // load a model, perform an action, pass the returned data to a variable
        // NOTE: please write the name of the model "LikeThis"
        $books_model = $this->loadModel('BooksModel');
        $books = $books_model->getAllBooks();

        // load views. Also, within the views we can echo out $books
        require 'application/view/header.php';
        require 'application/view/books/index.php';
        require 'application/view/footer.php';
    }

    /*
     * ACTION: addBook
```

```
     * This method handles what happens when you move to

     * http://localhost/mymvc/books/addbook

     * IMPORTANT: This is not a normal page, it's an ACTION.

     * This is where the "add a book" form on books/index

     * directs the user after the form submit. This method

     * handles all the POST data from the form and then redirects

     * the user back to books/index via the last line: header(...)

     * This is an example of how to handle a POST request.

     */

    public function addBook()

    {

        // simple message to show where you are

        echo 'Controller says: You are in the Controller <strong>Books</strong>, using the method
addBook().';



        // if we have POST data to create a new book entry

        if (isset($_POST["submit_add_book"])) {

            // load model, perform an action on the model

            $books_model = $this->loadModel('BooksModel');

            $books_model->addBook($_POST["title"],      $_POST["author"],      $_POST["publisher"],
$_POST["year"]);

        }



        // where to go after the book has been added

        header('location: ' . URL . 'books/index');

    }



    /*

     * ACTION: deleteBook

     * This method handles what happens when you move to

     * http://localhost/mymvc/books/deletebook

     * IMPORTANT: This is not a normal page, it's an ACTION.
```

```
     * This is where the "delete a book" button on books/index

     * directs the user after the click. This method handles

     * all the data from the GET request (in the URL) and then

     * redirects the user back to books/index via the last line: header(...)

     * This is an example of how to handle a GET request.

     * param int $book_id Id of the book to delete

     */

    public function deleteBook($book_id)

    {

        // simple message to show where you are

        echo 'Controller says: You are in the Controller <strong>Books</strong>, using the method
deleteBook().';



        // if we have an id of a book that should be deleted

        if (isset($book_id)) {

            // load model, perform an action on the model

            $books_model = $this->loadModel('BooksModel');

            $books_model->deleteBook($book_id);

        }



        // where to go after the book has been deleted

        header('location: ' . URL . 'books/index');

    }

}


?>
```

CREATING AND ADDING SOME VIEWS

Next, we will create the views in the "**/var/www/html/mymvc/application/view/**" folder. The view files, "**header.php**" and "**footer.php**" within this directory. The view files, "**index.php**", "**login.php**" and "**about.us**" within the subfolder "**home**" as these views are for the **home** controller.

../view/header.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>MVC skeleton</title>
    <meta name="description" content="MVC skeleton">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- css -->
    <link href="<?php echo URL; ?>public/css/style.css" rel="stylesheet">
    <!-- jQuery -->
    <script src="http://code.jquery.com/jquery-2.0.3.min.js"></script>
    <!-- our JavaScript -->
    <script src="<?php echo URL; ?>public/js/application.js"></script>
</head>
<body>
<!-- header -->
<div class="container">
    <!-- Info -->
    <div class="where-are-we-box">
        Everything in this box is loaded from <span class="bold">application/view/header.php</span>
        <br />
        The green line is added via JavaScript (to show how to integrate JavaScript).
    </div>
    <h1>The header (used on all pages)</h1>
    <!-- demo image -->
    <h3>Demo image, to show usage of public/img folder</h3>
    <div>
        <img src="<?php echo URL; ?>public/img/image.png" />
    </div>
    <!-- navigation -->
```

```
    <h3>Demo Navigation</h3>

    <div class="navigation">

        <ul>

            <!-- same like "home" or "home/index" -->

            <li><a href="<?php echo URL; ?>"><?php echo URL; ?>home</a></li>

            <li><a href="<?php echo URL; ?>home/login"><?php echo URL; ?>home/login</a></li>

            <li><a href="<?php echo URL; ?>home/aboutus"><?php echo URL; ?>home/aboutus</a></li>

        </ul>

    </div>

    <!-- simple div for javascript output, just to show how to integrate js into this MVC construct
-->

    <h3>Demo JavaScript</h3>

    <div id="javascript-header-demo-box">

    </div>

</div>
```

../view/footer.php

```
</body>

</html>
```

../view/home/index.php

```
<div class="container">
    <h2>You are in the View: application/view/home/index.php</h2>
    <p>You are in <strong>Home</strong></p>
</div>
```

../view/home/login.php

```
<div class="container">
    <h2>You are in the View: application/view/home/login.php</h2>
    <p>You are in <strong>Login</strong></p>
</div>
```

../view/home/aboutus.php

```
<div class="container">
    <h2>You are in the View: application/view/home/aboutus.php</h2>
    <p>You are in <strong>About Us</strong></p>
</div>
```

Let us create another view named “books”. For that, we will create the view folder

named “**books**” within the “**/var/www/html/mymvc/applcation/view/**” folder. The index.php with the contents as shown below, is within this folder containing the view for the controller named **books**, that we had created previously.

```
<div class="container">

    <h2>You are in the View: application/views/song/index.php</h2>

    <!-- add book form -->

    <div>

        <h3>Add a Book</h3>

        <form action="<?php echo URL; ?>books/addbook" method="POST">

            <label>Title</label>

            <input type="text" name="title" value="" required />

            <label>Author</label>

            <input type="text" name="author" value="" required />

            <label>Publisher</label>

            <input type="text" name="publisher" value="" />

            <label>Year</label>

            <input type="text" name="year" value="" />

            <input type="submit" name="submit_add_book" value="Submit" />

        </form>

    </div>

    <!-- main content output -->

    <div>

        <h3>List of books (data from our first model)</h3>

        <table>

            <thead style="background-color: #ddd; font-weight: bold;">

            <tr>

                  <td>Book id</td>

                <td>Book Title</td>

                <td>Author</td>

                <td>Publisher</td>

                <td>Year</td>
```

```
                <td> </td>

            </tr>

            </thead>

            <tbody>

            <?php foreach ($books as $book) { ?>

                <tr>

                    <td><?php if (isset($book->id)) echo $book->id; ?></td>

                    <td><?php if (isset($book->title)) echo $book->title; ?></td>

                    <td><?php if (isset($book->author)) echo $book->author; ?></td>

                    <td><?php if (isset($book->publisher)) echo $book->publisher; ?></td>

                    <td><?php if (isset($book->year)) echo $book->year; ?></td>

                    <td><a  href="<?php  echo  URL  .  'books/deletebook/'  .  $book->id;  ?>">DELETE
BOOK</a></td>

                </tr>

            <?php } ?>

            </tbody>

        </table>

    </div>

</div>
```

We can also decide to put all our ".css" ".js" and image files, that will be used in our views, to reside within the "**/var/www/html/mymvc/view/**" folder. That way we know where all our view related files are located.

CREATING AND ADDING A MODEL

Now, we need to create a model, which will perform some the database functions for our controller named **books,** that we had created earlier. Let us name our model file "**booksmodel.php**", populate it with the contents as shown below and save it within the folder "**/var/www/html/mymvc/application/model/**".

booksmodel.php

```
<?php



/*
```

```
 * Class Books

 *

 * Please note:

 * Don't use the same name for class and method,

 * as this might trigger an (unintended) __construct

 * of the class.

 * This is really weird behaviour, but documented here:

 * http://php.net/manual/en/language.oop5.decon.php

 *

 */

class Books extends Controller

{

    /*

     * PAGE: index

     * This method handles what happens when you move to

     * http://localhost/mymvc/books/index

     */

    public function index()

    {

        // simple message to show where you are

        echo 'Controller says: You are in the Controller <strong>Books</strong>, using the method
index().';



        // load a model, perform an action, pass the returned data to a variable

        // NOTE: please write the name of the model "LikeThis"

        $books_model = $this->loadModel('BooksModel');

        $books = $books_model->getAllBooks();



        // load views. Also, within the views we can echo out $books

        require 'application/view/header.php';

        require 'application/view/books/index.php';
```

```
        require 'application/view/footer.php';
    }

    /*
     * ACTION: addBook
     * This method handles what happens when you move to
     * http://localhost/mymvc/books/addbook
     * IMPORTANT: This is not a normal page, it's an ACTION.
     * This is where the "add a book" form on books/index
     * directs the user after the form submit. This method
     * handles all the POST data from the form and then redirects
     * the user back to books/index via the last line: header(...)
     * This is an example of how to handle a POST request.
     */
    public function addBook()
    {
        // simple message to show where you are
        echo 'Controller says: You are in the Controller <strong>Books</strong>, using the method
addBook().';

        // if we have POST data to create a new book entry
        if (isset($_POST["submit_add_book"])) {
            // load model, perform an action on the model
            $books_model = $this->loadModel('BooksModel');
            $books_model->addBook($_POST["title"],      $_POST["author"],      $_POST["publisher"],
$_POST["year"]);
        }

        // where to go after the book has been added
        header('location: ' . URL . 'books/index');
    }
```

```
    /*
     * ACTION: deleteBook
     * This method handles what happens when you move to
     * http://localhost/mymvc/books/deletebook
     * IMPORTANT: This is not a normal page, it's an ACTION.
     * This is where the "delete a book" button on books/index
     * directs the user after the click. This method handles
     * all the data from the GET request (in the URL) and then
     * redirects the user back to books/index via the last line: header(...)
     * This is an example of how to handle a GET request.
     * param int $book_id Id of the book to delete
     */
    public function deleteBook($book_id)
    {
        // simple message to show where you are
        echo 'Controller says: You are in the Controller <strong>Books</strong>, using the method
deleteBook().';

        // if we have an id of a book that should be deleted
        if (isset($book_id)) {
            // load model, perform an action on the model
            $books_model = $this->loadModel('BooksModel');
            $books_model->deleteBook($book_id);
        }

        // where to go after the book has been deleted
        header('location: ' . URL . 'books/index');
    }
}
?>
```

In the next section we will extend this barebone MVC called “mymvc” created in

this section to include our Login Form along with the styling elements.

7.6 BUILDING A PHP APPLICATION ON OUR MVC FRAMEWORK

In the previous section, we have build ourselves a MVC framework on which we will build our PHP application. However, we will make some modifications and additions to the MVC framework we created in the previous section, as we need to add styling and login functionality. Most of the PHP code contains comments for the ease of code readability and therefore we will go ahead and describe the final directory and file layout of the MVC application and list all the code files with the respective contents in this section. The code has been tested and should work well on any system, provided the previously mentioned software and configuration prerequisites are met.

THE FINAL DIRECTORY/FILE TREE

```
/var/www/html/mymvc/
├── application
│   ├── config
│   │   └── config.php
│   ├── controller
│   │   ├── books.php
│   │   └── home.php
│   ├── lib
│   │   ├── application.php
│   │   └── controller.php
│   ├── model
│   │   ├── booksmodel.php
│   │   └── loginmodel.php
```

```
|   └── view
|       ├── books
|       |   └── index.php
|       ├── css
|       |   └── mystyle.css
|       ├── footer.php
|       ├── header.php
|       ├── home
|       |   ├── aboutus.php
|       |   ├── index.php
|       |   └── login.php
|       ├── images
|       |   └── bg.gif
|       └── js
|           └── myjavascripts.js
├── .htaccess
└── index.php
```

CONTENTS OF THE .htaccess FILE

```
# To prevent problems when using a controller named "index" and having a root index.php
# more here: http://httpd.apache.org/docs/2.2/content-negotiation.html
Options -MultiViews

# Activates URL rewriting (like localhost/controller/action/1/2/3)
RewriteEngine On

# Disallows others to look directly into /folder/
Options -Indexes
```

```
# When using the script within a sub-folder, put this path here, like /mysubfolder/

# If your app is in the root of your web folder, then leave it commented out

RewriteBase /mymvc/


# General rewrite rules

RewriteCond %{REQUEST_FILENAME} !-d

RewriteCond %{REQUEST_FILENAME} !-f

RewriteCond %{REQUEST_FILENAME} !-l



RewriteRule ^(.+)$ index.php?url=$1 [QSA,L]
```

CONTENTS OF THE index.php FILE

```
<?php

// start the session

session_start();

// set time-out period (in seconds)

$inactive = 60;



// check to see if $_SESSION["timeout"] is set

if (isset($_SESSION["timeout"])) {

    // calculate the session's "time to live"

    $sessionTTL = time() - $_SESSION["timeout"];

    if ($sessionTTL > $inactive) {

        session_destroy();

       header('location: ' . URL . 'home/logmeout');

    }

}
```

```
$_SESSION["timeout"] = time();



/*
 * A simple MVC skeleton
 */


// load application config (error reporting etc.)
require 'application/config/config.php';


// load application class
require 'application/lib/application.php';
require 'application/lib/controller.php';


// start the application
$app = new Application();


?>
```

CONTENTS OF THE application/config/config.php FILE

```
<?php


/*
 * Configuration
 *
```

```
 * See http://php.net/manual/en/function.define.php
 */


/*
 * Configuration for: Error reporting
 * Useful to show every little problem during development
 */
error_reporting(E_ALL);
ini_set("display_errors", 1);


/*
 * Configuration for: Project URL
 * Put your URL here, "127.0.0.1" or "localhost"
 */
define('URL', 'http://localhost/mymvc/');


/*
 * Configuration for: Database
 * Define your database credentials, database type etc.
 */
define('DB_TYPE', 'mysql');
define('DB_HOST', '127.0.0.1');
define('DB_NAME', 'mymvc');
define('DB_USER', 'root');
define('DB_PASS', 'dbpassword');
```

```
/*
 * Salt to enhance password security
 */
define('MY_SALT', '@kKh|+-|SoU%');

?>
```

CONTENTS OF THE application/controller/books.php FILE

```
<?php

/*
 * Class Books
 *
 * Please note:
 * Don't use the same name for class and method,
 * as this might trigger an (unintended) __construct
 * of the class.
 * This is really weird behaviour, but documented here:
 * http://php.net/manual/en/language.oop5.decon.php
 *
 */
class Books extends Controller
{
    /*
```

```
     * PAGE: index

     * This method handles what happens when you move to

     * http://localhost/mymvc/books/index

     */

    public function index()

    {

        // simple message to show where you are

        echo 'Controller says: You are in the Controller
<strong>Books</strong>, using the method index().';



        // load a model, perform an action, pass the returned data to a
variable

        // NOTE: please write the name of the model "LikeThis"

        $books_model = $this->loadModel('BooksModel');

        $books = $books_model->getAllBooks();



        // load views. Also, within the views we can echo out $books

        require 'application/view/header.php';

        require 'application/view/books/index.php';

        require 'application/view/footer.php';

    }



    /*

     * ACTION: addBook

     * This method handles what happens when you move to

     * http://localhost/mymvc/books/addbook
```

```
     * IMPORTANT: This is not a normal page, it's an ACTION.
     * This is where the "add a book" form on books/index
     * directs the user after the form submit. This method
     * handles all the POST data from the form and then redirects
     * the user back to books/index via the last line: header(...)
     * This is an example of how to handle a POST request.
     */
    public function addBook()
    {
        // simple message to show where you are
        echo 'Controller says: You are in the Controller <strong>Books</strong>, using the method addBook().';

        // if we have POST data to create a new book entry
        if (isset($_POST["submit_add_book"])) {
            // load model, perform an action on the model
            $books_model = $this->loadModel('BooksModel');
            $books_model->addBook($_POST["title"], $_POST["author"], $_POST["publisher"], $_POST["year"]);
        }

        // where to go after the book has been added
        header('location: ' . URL . 'books/index');
    }
```

```
    /*
     * ACTION: deleteBook
     * This method handles what happens when you move to
     * http://localhost/mymvc/books/deletebook
     * IMPORTANT: This is not a normal page, it's an ACTION.
     * This is where the "delete a book" button on books/index
     * directs the user after the click. This method handles
     * all the data from the GET request (in the URL) and then
     * redirects the user back to books/index via the last line: header(...)
     * This is an example of how to handle a GET request.
     * param int $book_id Id of the book to delete
     */
    public function deleteBook($book_id)
    {
        // simple message to show where you are
        echo 'Controller says: You are in the Controller
<strong>Books</strong>, using the method deleteBook().';

        // if we have an id of a book that should be deleted
        if (isset($book_id)) {
            // load model, perform an action on the model
            $books_model = $this->loadModel('BooksModel');
            $books_model->deleteBook($book_id);
        }
```

```
        // where to go after the book has been deleted
        header('location: ' . URL . 'books/index');
    }
}
?>
```

CONTENTS OF THE application/controller/home.php FILE

```
<?php

/*
 * Class Home
 *
 * Please note:
 * Don't use the same name for class and method,
 * as this might trigger an (unintended) __construct
 * of the class. This is really weird behaviour, but
 * documented here:
 * http://php.net/manual/en/language.oop5.decon.php
 *
 */
class Home extends Controller
{
    /*
     * PAGE: index
     * This method handles what happens when you move
     * to http://yourproject/home/index (which is the
```

```
     * default page)
     */
    public function index()
    {
        // debug message to show where you are, just for the demo
        echo 'Controller says: you are in the controller
<strong>home</strong>, using the method <em>index()</em>.';
        // load some views
        require 'application/view/header.php';
        require 'application/view/home/index.php';
        require 'application/view/footer.php';
    }

    /*
     * PAGE: login
     * This method handles what happens when you move to
     * http://localhost/home/login
     * The camelCase writing is just for better readability.
     * The method name is case insensitive.
     */
    public function logIn()
    {
        // debug message to show where you are, just for the demo
        echo 'Controller says: you are in the controller
<strong>home</strong>, using the method <em>login()</em>.';
```

```
        // load some views
        require 'application/view/header.php';
        require 'application/view/home/login.php';
        require 'application/view/footer.php';
    }

    /*
     * ACTION: logmein
     * This method handles what happens when you move to
     * http://localhost/mymvc/home/logmein
     * IMPORTANT: This is not a normal page, it's an ACTION.
     * This is where the "Login" button on the Login Form
     * directs the user after the login form submit. This method
     * handles all the POST data from the form and then redirects
     * the user to books/index via the last line: header(...)
     */
    public function logMeIn()
    {
        // simple message to show where you are
        echo 'Controller says: you are in the Controller
<strong>home</strong>, using the method <em>logmein()</em>.';

        // if we have POST data to login
        if (isset($_POST["submit_login"])) {
```

```
            // load model, perform an action on the model

            $login_model = $this->loadModel('LoginModel');



           // check username and password combination

           if                      ($login_model->checkUser($_POST["username"],
$_POST["password"])) {



                // set the session variables

                $_SESSION["username"] = $_POST["username"];

                $_SESSION["loggedin"] = TRUE;



              // username password correct

              header('location: ' . URL . 'books/index');



           } else {



              // username password incorrect

              echo '<p><font color="#ff0000">Please check your username and
password and try again</font></p>';

               // load some views

              require 'application/view/header.php';

               require 'application/view/home/login.php';

              require 'application/view/footer.php';

           }



       }
```

```
    }

    /*
     * ACTION: logmeout
     * This method handles what happens when you move to
     * http://localhost/mymvc/home/logmeout
     * IMPORTANT: This is not a normal page, it's an ACTION.
     * This is where the "Logout" button on the Login Page
     * directs the user to the login form page. This method
     * handles resets all the session variables and then redirects
     * the user to home/index via the last line: header(...)
     */
    public function logMeOut()
    {
        // simple message to show where you are
        echo 'Controller says: you are in the Controller <strong>home</strong>, using the method <em>logmeout()</em>.';

        // delete the username, loggedin values
        unset($_SESSION["username"]);
        unset($_SESSION["loggedin"]);

        // unset all session values
        session_unset();
```

```
        // destroy the session
        session_destroy();

       // redirect to the default index page
        header('location: ' . URL . 'home/index');
    }



    /*
     * PAGE: aboutus
     * This method handles what happens when you move to
     * http://localhost/home/aboutus
     * The camelCase writing is just for better readability.
     * The method name is case insensitive.
     */
    public function aboutUs()
    {
        // debug message to show where you are, just for the demo
        echo 'Controller says: you are in the controller
<strong>home</strong>, using the method <em>aboutus()</em>.';
        // load some views
        require 'application/view/header.php';
        require 'application/view/home/aboutus.php';
        require 'application/view/footer.php';
```

```
    }


}


?>
```

CONTENTS OF THE application/lib/application.php FILE

```
<?php


class Application

{

    /* The controller set to null */

    private $url_controller = null;


    /* The method (of the above controller) set to null */

    private $url_action = null;


    /* Set the Parameter one to null */

    private $url_parameter_1 = null;


    /* Set the Parameter two to null */

    private $url_parameter_2 = null;


    /* Set the Parameter three to null */

    private $url_parameter_3 = null;
```

```
    /*
     * "Start" the application:
     * Analyzes the URL elements and calls accordingly
     * the controller/method or the fallback
     */
    public function __construct()
    {
        // create array with URL parts in $url
        $this->splitUrl();

        // check controller: does such a controller exist ?
        if (file_exists('./application/controller/' . $this->url_controller .
'.php')) {

            // if so, then load this file and create this controller
            // example: if controller would be "login", then this line
           // would translate into: $this->login = new login();
            require  './application/controller/'  .  $this->url_controller  .
'.php';
            $this->url_controller = new $this->url_controller();

            // check method: does such a method exist in the controller ?
            if (method_exists($this->url_controller, $this->url_action)) {

                // call the method and pass the arguments to it
```

```
                if (isset($this->url_parameter_3)) {

                    // will translate to something like $this->home->method($param_1, $param_2, $param_3);

                    $this->url_controller->{$this->url_action}($this->url_parameter_1, $this->url_parameter_2, $this->url_parameter_3);

                } elseif (isset($this->url_parameter_2)) {

                    // will translate to something like $this->home->method($param_1, $param_2);

                    $this->url_controller->{$this->url_action}($this->url_parameter_1, $this->url_parameter_2);

                } elseif (isset($this->url_parameter_1)) {

                    // will translate to something like $this->home->method($param_1);

                    $this->url_controller->{$this->url_action}($this->url_parameter_1);

                } else {

                    // if no parameters given, just call the method without parameters, like $this->home->method();

                    $this->url_controller->{$this->url_action}();

                }

            } else {

                // default/fallback: call the index() method of a selected controller

                $this->url_controller->index();

            }

        } else {

            // invalid URL, so simply show home/index

            require './application/controller/home.php';
```

```
            $home = new Home();
            $home->index();
        }
    }

    /*
     * Get and split the URL
     */
    private function splitUrl()
    {
        if (isset($_GET['url'])) {

            // split URL
            $url = rtrim($_GET['url'], '/');
            $url = filter_var($url, FILTER_SANITIZE_URL);
            $url = explode('/', $url);

            // Put URL parts into according properties
            // By the way, the syntax here is just a short form of if/else,
called "Ternary Operators"
            //    @see    http://davidwalsh.name/php-shorthand-if-else-ternary-
operators
            $this->url_controller = (isset($url[0]) ? $url[0] : null);
            $this->url_action = (isset($url[1]) ? $url[1] : null);
            $this->url_parameter_1 = (isset($url[2]) ? $url[2] : null);
            $this->url_parameter_2 = (isset($url[3]) ? $url[3] : null);
```

```
            $this->url_parameter_3 = (isset($url[4]) ? $url[4] : null);


            // for debugging. uncomment this if you have problems with the URL
            // echo 'Controller: ' . $this->url_controller . '<br />';
            // echo 'Action: ' . $this->url_action . '<br />';
            // echo 'Parameter 1: ' . $this->url_parameter_1 . '<br />';
            // echo 'Parameter 2: ' . $this->url_parameter_2 . '<br />';
            // echo 'Parameter 3: ' . $this->url_parameter_3 . '<br />';
        }
    }
}
?>
```

CONTENTS OF THE application/lib/controller.php FILE

```
<?php


/*
 * This is the "base controller class".
 * All other "real" controllers extend this class.
 */
class Controller
{
    /*
     * The Database Connection is set to null
     */
```

```
    public $db = null;


    /*
     * Whenever a controller is created, open
     * a database connection too. The idea behind
     * this is to have ONE connection that can be
     * used by multiple models (there are frameworks
     * that open one connection per model).
     */
    function __construct()
    {
        $this->openDatabaseConnection();
    }


    /*
     * Open the database connection with the credentials
     * from the application/config/config.php file
     */
    private function openDatabaseConnection()
    {
        // set the (optional) options of the PDO connection.
        // In this case, we set the fetch mode to "objects",
        // which means all results will be objects, like this:
        // $result->username
        // For example, the fetch mode FETCH_ASSOC would return
```

```
        // results like this: $result["username"]

        // see http://www.php.net/manual/en/pdostatement.fetch.php

        $options = array(PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ,
PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING);



        // generate a database connection and using the PDO connector

        $this->db = new PDO(DB_TYPE . ':host=' . DB_HOST . ';dbname=' .
DB_NAME, DB_USER, DB_PASS, $options);

    }



    /*

     * Load the model with the given name.

     * loadModel("BooksModel") would include models/booksmodel.php

     * and create the object in the controller, like this:

     * $books_model = $this->loadModel('BooksModel');

     * Note that the model class name is written in "CamelCase",

     * the model's filename is the same in lowercase letters

     * param string $model_name The name of the model

     * return object model

     */

    public function loadModel($model_name)

    {

        require 'application/model/' . strtolower($model_name) . '.php';

        // return new model (and pass the database connection to the model)

        return new $model_name($this->db);
```

```
    }
}
?>
```

CONTENTS OF THE application/model/booksmodel.php FILE

```
<?php


class BooksModel
{
    /*
     * Every model needs a database connection, passed to the model
     * param object $db is a PDO database connection
     */
    function __construct($db) {
        try {
            $this->db = $db;
        } catch (PDOException $e) {
            exit('Database connection could not be established.');
        }
    }


    /*
     * Get all the books from the database
     */
    public function getAllBooks()
    {
```

```
        $sql = "SELECT id, title, author, publisher, year FROM books";
        $query = $this->db->prepare($sql);
        $query->execute();

        // fetchAll() is the PDO method that gets all result rows,
       // here in object-style because we defined this in the file
        // lib/controller.php. If you prefer to get an associative
       // array as the result, then $query->fetchAll(PDO::FETCH_ASSOC);
       // or change in the lib/controller.php's PDO options to
        //  $options = array(PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
...
        return $query->fetchAll();
    }

    /*
     * Add a book to the database
     * param string $title Title
     * param string $author Author
     * param string $publisher Publisher
     * param string $year Year
     */
    public function addBook($title, $author, $publisher, $year)
    {
        // clean the input from javascript code for example
        $title = strip_tags($title);
```

```
        $author = strip_tags($author);

        $publisher = strip_tags($publisher);

        $year = strip_tags($year);



        $sql = "INSERT INTO books (title, author, publisher, year) VALUES
(:title, :author, :publisher, :year)";

        $query = $this->db->prepare($sql);

        $query->execute(array(':title'  =>  $title,  ':author'  =>  $author,
':publisher' => $publisher, ':year' => $year));

    }



    /*

     * Delete a book from the database

     * param int $book_id Id of a Book

     */

    public function deleteBook($book_id)

    {

        $sql = "DELETE FROM books WHERE id = :book_id";

        $query = $this->db->prepare($sql);

        $query->execute(array(':book_id' => $book_id));

    }

}

?>
```

CONTENTS OF THE application/model/loginmodel.php FILE

```
<?php
```

```
class LoginModel

{

    /*

     * Every model needs a database connection, passed to the model

     * param object $db is a PDO database connection

     */

    function __construct($db) {

        try {

            $this->db = $db;

        } catch (PDOException $e) {

            exit('Database connection could not be established.');

        }

    }



    /*

     * Check the username and password in the database

     * param string $username Login Username

     * param string $password Login Password

     */

    public function checkUser($username, $password)

    {

        $username = strip_tags($username);

        $password = strip_tags($password);



        $sql = "SELECT username, password FROM `users` WHERE username =
```

```
'".$username."' AND password = md5('\"".$password."\" . \"".MY_SALT."\"')";

        $query = $this->db->prepare($sql);

       $query->execute();

       $numrows = $query->rowCount();



        // fetchAll() is the PDO method that gets all result rows,

        // here in object-style because we defined this in the file

        // lib/controller.php. If you prefer to get an associative

        // array as the result, then $query->fetchAll(PDO::FETCH_ASSOC);

        // or change in the lib/controller.php's PDO options to

        //  $options  =  array(PDO::ATTR_DEFAULT_FETCH_MODE  =>  PDO::FETCH_ASSOC
...

        if ($numrows > 0) {

              // getting to this point means that the credentials match

              return TRUE;

       } else {

              // getting to this points means unsuccessful login

              return FALSE;

       }

    }


}


?>
```

CONTENTS OF THE application/view/books/index.php FILE

```
<?php



 // check and see if the user is logged in

 if (isset($_SESSION["loggedin"])) {



       // user is logged in show the Page

       echo  '<p>Hey  you  have  logged  in  with  the  username  <strong>'  .
$_SESSION["username"] . '</strong></p>';

?>

<div class="container">

    <p>You              are              in              the              View:
"<strong>application/views/song/index.php</strong>"</p>

    <!-- add book form -->

    <div>

        <h3>Add a Book</h3>

        <form action="<?php echo URL; ?>books/addbook" method="POST">

            <label>Title</label>

            <input type="text" name="title" value="" required />

            <label>Author</label>

            <input type="text" name="author" value="" required />

            <label>Publisher</label>

            <input type="text" name="publisher" value="" required/>

            <label>Year</label>

            <input type="text" name="year" value="" required/>

            <input type="submit" name="submit_add_book" value="Submit" />

        </form>
```

```
    </div>
    <!-- main content output -->
    <div>
        <h3>List of books (data from our first model)</h3>
        <table>
            <thead style="background-color: #ddd; font-weight: bold;">
            <tr>
              <td>Book id</td>
                <td>Book Title</td>
                <td>Author</td>
                <td>Publisher</td>
                <td>Year</td>
                <td> </td>
            </tr>
            </thead>
            <tbody>
            <?php foreach ($books as $book) { ?>
                <tr>
                    <td><?php if (isset($book->id)) echo $book->id; ?></td>
                    <td><?php  if  (isset($book->title))  echo  $book->title;
?></td>
                    <td><?php  if  (isset($book->author))  echo  $book->author;
?></td>
                    <td><?php   if   (isset($book->publisher))   echo   $book-
>publisher; ?></td>
                    <td><?php   if   (isset($book->year))   echo   $book->year;
```

```
?></td>

                    <td><a href="<?php echo URL . 'books/deletebook/' . $book-
>id; ?>">DELETE BOOK</a></td>

                </tr>

            <?php } ?>

            </tbody>

        </table>

    </div>

</div>

<?php

 } else {

       // user is not logged in, send user to login page

       header('location: ' . URL . 'home/login');



}

?>
```

CONTENTS OF THE application/view/css/mystyle.css FILE

```
/* login-form class definition */

.login-form {

       color:darkred; /* darkred text colour */

       text-transform:uppercase; /* text in uppercase */

       border-style:dotted; /* dotted border */

       border-width:2px; /* 2 pixels border width */

       border-color:#ff0000;  /* red border colour */

       background-color:#cccccc; /* background colour is a shade of grey */

}
```

```
/* apply to all the td elements of the login-form class */
.login-form td {
    text-align:center; /* center align td elements */
    vertical-align:middle;  /* vertically middle align td elements */
}

/* apply to all the input elements of the login-form class */
.login-form input {
       color:darkred; /* input elements have darkred text colour */
       background-color:#eeeeee; /* input elements have grey background */
       height:25px; /* input elements have 25 pixels height */
       width:200px; /* input elements have 200 pixels width */
}

.view-box{
    height: 500px;
    width: 500px;
    position: relative;
    background-color: #FFFFFF;
    border-width: 1px;
    border-style: solid;
    border-color: #dddddd;
    border-radius: 0px;
```

```
    box-shadow: 0px 10px 6px -6px #777;

}



/* some styling for the menu */

ul.menu

{

    padding:0;

    list-style-type: none;

    height: 26px;

    /*width:500px;margin:0  auto;*//*Uncomment  this  line  to  make  the  menu
center-aligned.*/

}



ul.menu li

{

    border:1px solid #65A2DC;

    border-right:none;

    list-style-type: none;

    padding:0;margin:0;

    float:left;

    display: block;

    color:White;

}

ul.menu li.lastItem

{
```

```
    border-right:1px solid #65A2DC;
}

ul.menu li a
{
    padding: 0 20px;
    background-image: url(/mymvc/application/view/images/bg.gif);
    border:1px solid #DDECF9;
    border-top:1px solid #FFFFFF;
    color:White;
    text-align: left;
    text-decoration:none;
    display: block;
    float: left;
    font: bold 12px Arial;
    line-height: 26px;
}

ul.menu li a:hover, ul.menu li a.current
{
    background-position:0 -40px;
}

/* header */
.header {
```

```
  background: #efefef;
}

/* footer */
.footer {
  background: #eeeeee;
}
```

CONTENTS OF THE application/view/footer.php FILE

```
<div class="footer">
       <p>This is my footer</p>
</div>
</body>
</html>
```

CONTENTS OF THE application/view/header.php FILE

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>MVC skeleton</title>
    <meta name="description" content="MVC skeleton">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- css -->
    <link    href="<?php    echo    URL;    ?>application/view/css/mystyle.css"
```

```
rel="stylesheet">

    <!-- jQuery -->

    <script src="http://code.jquery.com/jquery-2.0.3.min.js"></script>

    <!-- our JavaScript -->

    <script                src="<?php                echo                URL;
?>application/view/js/myjavascripts.js"></script>

</head>

<body>

<!-- header -->

<div class="container">

    <div class="header">

       <p>This is my header</p>

    </div>

    <!-- navigation -->

    <div class="navigation">

        <ul class="menu">

            <!-- same like "home" or "home/index" -->

            <li><a href="<?php echo URL; ?>">Home</a></li>

            <li><a href="<?php echo URL; ?>home/login">Login</a></li>

            <li><a href="<?php echo URL; ?>home/aboutus">Aboutus</a></li>

            <li      class="lastItem"><a      href="<?php      echo      URL;
?>home/logmeout">Logout</a></li>

        </ul>

    </div>

</div>
```

CONTENTS OF THE application/view/home/aboutus.php FILE

```
<div class="view-box">

    <p>You              are              in              the              View:
"<strong>application/view/home/aboutus.php</strong>"</p>

    <p>You are in <strong>About Us</strong></p>

</div>
```

CONTENTS OF THE application/view/home/index.php FILE

```
<div class="view-box">

    <p>You              are              in              the              View:
"<strong>application/view/home/index.php</strong>"</p>

    <p>You are in <strong>Home</strong></p>

</div>
```

CONTENTS OF THE application/view/home/login.php FILE

```
<div class="view-box">

    <p>You              are              in              the              View:
"<strong>application/view/home/login.php</strong>"</p>

    <p>You are in <strong>Login</strong></p>

    <form action="<?php echo URL; ?>home/logmein" method="POST">

    <table class="login-form">

       <tr>

              <td>Message:</td><td><input   id="messageBox"   name="messageBox"
type="text" name="messageBox" readonly></td>

       </tr>

       <tr>

              <td>Username:</td><td><input    id="username"    name="username"
type="text" name="username" onkeyup="checkUsername()"></td>

       </tr>

       <tr>
```

```
            <td>Password:</td><td><input    id="password"    name="password"
type="password" name="password"></td>

       </tr>

       <tr>

            <td   colspan="2"><input   id="submit_login"   name="submit_login"
type="submit" value="Login"></td>

       </tr>

    </table>

    </form>

</div>
```

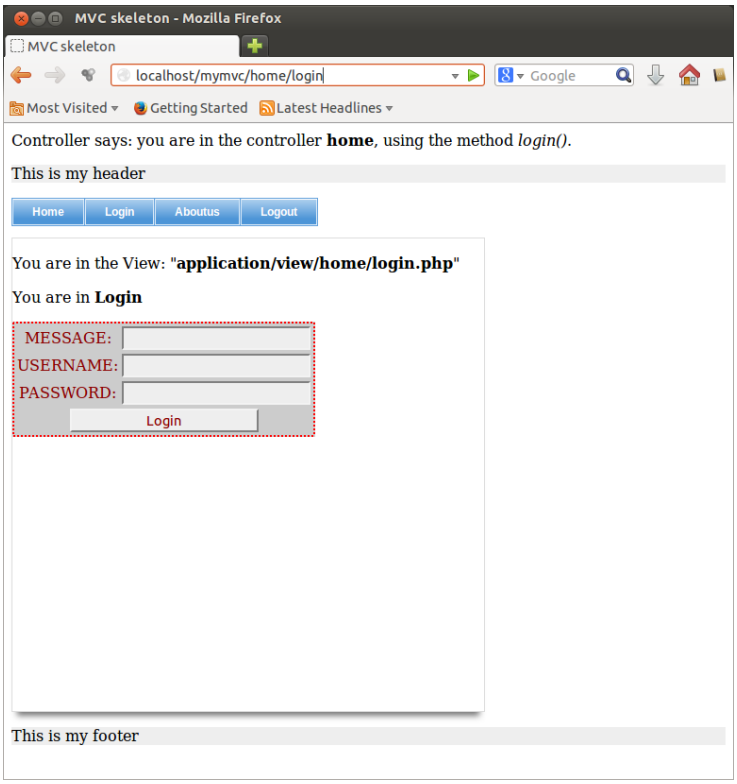
CONTENTS OF THE application/view/image/bg.gif FILE

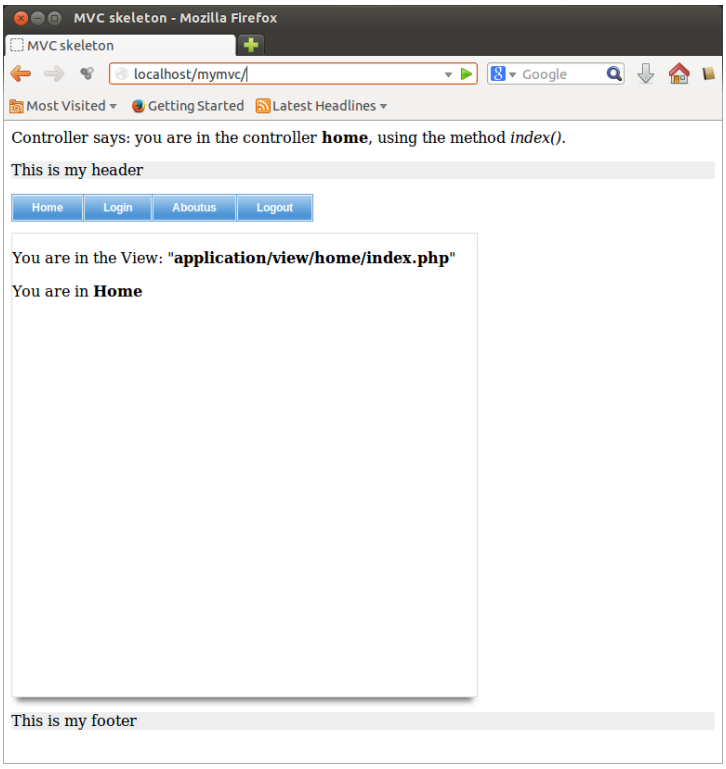
CONTENTS OF THE application/view/js/myjavascripts.js FILE

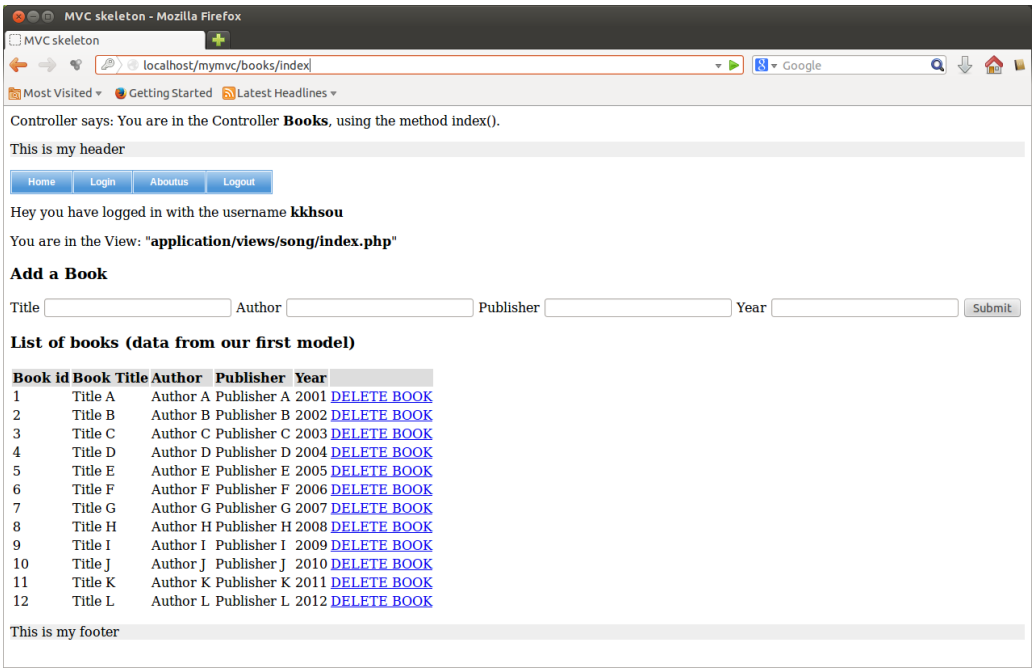
```
//function checkUsername checks the username input; whether blank,

//numbers, letters or special characters.

function checkUsername()

{

       var username;

       username = document.getElementById("username").value;



       if ( username=="" )

       {

              document.getElementById("messageBox").value = "Blank Username";
```

```
	}
	else if ( username.match(/[a-z]/i) )
	{
		document.getElementById("messageBox").value = "Alphabets";
	}
	else if ( username.match(/[0-9]/) )
	{
		document.getElementById("messageBox").value = "Numbers";
	}
	else
	{
		document.getElementById("messageBox").value = "Special
Characters";
	}
}
```

SOME SCREEN CAPTURES OF OUR MVC







7.7 LET US SUM UP

In this unit, we have briefly discussed the basics of the MVC design pattern. Though we have covered only a few of the topics, these topics are intended to inspire and point you in a direction to further explore. This unit was not intended to provide you with an exhaustive in-depth on the topics but merely to introduce you to some of the basic concepts of the MVC architecture and use them in building web pages.

What we have learned in this unit.

- MVC basics and Coding considerations
- Building a MVC framework
- Using our MVC framework to write PHP a web application

CHECK YOUR PROGRESS

Q1: MVC is

(a) a web page (b) a scripting language

(c) a framework (d) a markup language

Q2: In MVC, what does the **M** stand for ?

(a) Model (b) Markup

(c) Modern (d) Machine

Q3: What does the **C** stand for in MVC ?

(a) C language (b) Characteristics

(c) Controller (d) Compiler

Q4: MVC stands for which of the following ?

(a) Model View Controller (b) Modern View Compiler

(c) Machine View Controller (d) Machine View Compiler

Q5: Which of the following location contains the default **DocumentRoot** for the Apache Web Server?

(a) /var/www/html (b) /var/log/httpd

(c) /etc/httpd (d) /etc/www/html

Q6: Where is the main Apache Server configuration file located?

Q7: Which command can be used to change to the root user?

Q8: How can you assign variables by reference in PHP?

Q9: Which of the following data types are compound data types?

(a) boolean (b) string

(c) object (d) array

Q10: What does the PHP expression **$a === $b** mean?

7.8 FURTHER READINGS

- Chris Pitt, "Pro PHP MVC", Apress, ISBN-13 9781430241645, ISBN-10 1430241640, 2012
- W. Jason Gilmore, "Beginning PHP and MySQL: From Novice to Professional 4th Edition", Apress, ISBN-13 9788184897456, ISBN-10 8184897456, 2010
- Steven Holzner, "PHP: The Complete Reference 1st Edition", Tata Mcgraw Hill Education Private Limited, ISBN-13 9780070223622, ISBN-10 0070223629, 2007
- http://www.php.net/manual/en/
- http://en.wikipedia.org/wiki/PHP
- http://in2.php.net/FAQ.php

7.9 ANSWERS TO CHECK YOUR PROGRESS

1. (c)
2. (a)
3. (c)
4. (a)
5. (a)
6. /etc/httpd/conf/httpd.conf
7. su -
8. Assigning variables by reference is done by using the & operator. For example,
$my_var = &$my_variable

9. (c), (d)

10. Identical, TRUE if $a is equal to $b, and are of the same data type as well.

7.10 MODEL QUESTIONS

1. What is MVC ?
2. How can the methods of a class be used in PHP?
3. How do we test if the apache **mod_rewrite** module is available and working?
4. Which command is used to create a database in MySQL?
5. Which symbol/sign represents the start of a variable name in PHP?
6. Which MySQL command is used to insert values in tables?
7. Which Apache module is required for PHP scripts to connect to MySQL?
8. What does the Apache **mod_rewrite** module do?
9. Which are the start and end tags that the PHP parser reads to start parsing and stop parsing PHP code?
10. Which command is used to list the table contents in MySQL?

****