

Introduction to Web Programming

CSC 293



*University of Ibadan Distance Learning Centre
Open and Distance Learning Course Series Development
Version 1.0 v1*

Copyright © 2016 by Distance Learning Centre, University of Ibadan, Ibadan.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owner.

ISBN: 978-021-723-1

General Editor: Prof. Bayo Okunade

University of Ibadan Distance Learning Centre
University of Ibadan,
Nigeria

Telex: 31128NG

Tel: +234 (80775935727)

E-mail: ssu@dlc.ui.edu.ng

Website: www.dlc.ui.edu.ng

Vice-Chancellor's Message

The Distance Learning Centre is building on a solid tradition of over two decades of service in the provision of External Studies Programme and now Distance Learning Education in Nigeria and beyond. The Distance Learning mode to which we are committed is providing access to many deserving Nigerians in having access to higher education especially those who by the nature of their engagement do not have the luxury of full time education. Recently, it is contributing in no small measure to providing places for teeming Nigerian youths who for one reason or the other could not get admission into the conventional universities.

These course materials have been written by writers specially trained in ODL course delivery. The writers have made great efforts to provide up to date information, knowledge and skills in the different disciplines and ensure that the materials are user-friendly.

In addition to provision of course materials in print and e-format, a lot of Information Technology input has also gone into the deployment of course materials. Most of them can be downloaded from the DLC website and are available in audio format which you can also download into your mobile phones, IPod, MP3 among other devices to allow you listen to the audio study sessions. Some of the study session materials have been scripted and are being broadcast on the university's Diamond Radio FM 101.1, while others have been delivered and captured in audio-visual format in a classroom environment for use by our students. Detailed information on availability and access is available on the website. We will continue in our efforts to provide and review course materials for our courses.

However, for you to take advantage of these formats, you will need to improve on your I.T. skills and develop requisite distance learning Culture. It is well known that, for efficient and effective provision of Distance learning education, availability of appropriate and relevant course materials is a *sine qua non*. So also, is the availability of multiple plat form for the convenience of our students. It is in fulfilment of this, that series of course materials are being written to enable our students study at their own pace and convenience.

It is our hope that you will put these course materials to the best use.



Prof. Abel Idowu Olayinka

Vice-Chancellor

Foreword

As part of its vision of providing education for “Liberty and Development” for Nigerians and the International Community, the University of Ibadan, Distance Learning Centre has recently embarked on a vigorous repositioning agenda which aimed at embracing a holistic and all encompassing approach to the delivery of its Open Distance Learning (ODL) programmes. Thus we are committed to global best practices in distance learning provision. Apart from providing an efficient administrative and academic support for our students, we are committed to providing educational resource materials for the use of our students. We are convinced that, without an up-to-date, learner-friendly and distance learning compliant course materials, there cannot be any basis to lay claim to being a provider of distance learning education. Indeed, availability of appropriate course materials in multiple formats is the hub of any distance learning provision worldwide.

In view of the above, we are vigorously pursuing as a matter of priority, the provision of credible, learner-friendly and interactive course materials for all our courses. We commissioned the authoring of, and review of course materials to teams of experts and their outputs were subjected to rigorous peer review to ensure standard. The approach not only emphasizes cognitive knowledge, but also skills and humane values which are at the core of education, even in an ICT age.

The development of the materials which is on-going also had input from experienced editors and illustrators who have ensured that they are accurate, current and learner-friendly. They are specially written with distance learners in mind. This is very important because, distance learning involves non-residential students who can often feel isolated from the community of learners.

It is important to note that, for a distance learner to excel there is the need to source and read relevant materials apart from this course material. Therefore, adequate supplementary reading materials as well as other information sources are suggested in the course materials.

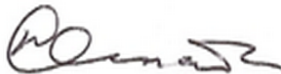
Apart from the responsibility for you to read this course material with others, you are also advised to seek assistance from your course facilitators especially academic advisors during your study even before the interactive session which is by design for revision. Your academic advisors will assist you using convenient technology including Google Hang Out, You Tube, Talk Fusion, etc. but you have to take advantage of these. It is also going to be of immense advantage if you complete assignments as at when due so as to have necessary feedbacks as a guide.

The implication of the above is that, a distance learner has a responsibility to develop requisite distance learning culture which includes diligent and disciplined self-study, seeking available administrative and academic support and acquisition of basic information technology skills. This is why you are encouraged to develop your computer skills by availing yourself the opportunity of training that the Centre's provide and put these into use.

In conclusion, it is envisaged that the course materials would also be useful for the regular students of tertiary institutions in Nigeria who are faced with a dearth of high quality textbooks. We are therefore, delighted to present these titles to both our distance learning students and the university's regular students. We are confident that the materials will be an invaluable resource to all.

We would like to thank all our authors, reviewers and production staff for the high quality of work.

Best wishes.

A handwritten signature in black ink, appearing to read 'Okunade', written in a cursive style.

Professor Bayo Okunade

Course Development Team

Content Authoring

Yetunde Folajimi

Content Editor

Prof. Remi Raji-Oyelade

Production Editor

Ogundele Olumuyiwa Caleb

Learning Design/Assessment Authoring

Folajimi Olambo Fakoya

Managing Editor

Ogunmefun Oladele Abiodun

General Editor

Prof. Bayo Okunade

Contents

About this course manual	8
How this course manual is structured.....	8
Getting around this course manual	10
Margin icons.....	10
Study Session 1	11
Understanding the World Wide Web	11
Introduction	11
Terminology.....	11
1.1 Web Browsers.....	12
1.2 Web Usability	12
1.2.1 Designing Web Pages	12
1.3 Web site Organization	13
1.3.1 File Naming Conventions	13
1.4 Site Structure.....	13
1.5 Web Editors	14
1.5.1 Adobe Dreamweaver	14
1.5.2 Notepad++	14
1.5.3 Bluefish	14
1.6 Evaluating Your Web site.....	14
1.6.1 Designing Your Page to Maximize Rankings.....	15
1.7 Moving Your Files to a Web Server.....	16
1.7.1 Uniform Resource Identifiers (URIs)	16
1.8 What a URL Is Made Up of	16
1.8.1 The Scheme	17
1.8.2 The Host Address	17
1.8.3 The File path	18
1.8.4 Other Parts of the URL.....	18
1.8.4.1 Ports.....	18
1.8.4.2 Fragment Identifiers	19
1.9 Absolute and Relative URLs.....	19
1.9.1 Different Types of Relative URLs.....	20
1.9.1.1 Same Directory.....	20
1.9.1.2 Subdirectory.....	20
1.9.1.3 Parent Directory	21
1.9.1.4 From the Root.....	21
1.9.1.5 Default Files	21
1.10 The Hypertext Transfer Protocol.....	21

Study Session Summary	22
Assessment.....	22

Study Session 2 24

Web Programming	24
Expected duration: 1 week or 2 contact hour	24
Introduction	24
Terminology.....	24
2.1 Web Programming.....	25
2.2 Client-side Versus Server-side Scripting.....	25
2.2.1 Client Side Scripting	26
2.2.2 Client-side Environment	26
2.2.3 Server side Scripting	27
2.2.3 Server-side Environment.....	27
2.3 Deployment and platform.....	28
2.4 State and secondary effects.....	28
Study Session Summary	28
Assessment.....	29

Study Session 3 29

Web Application	29
Introduction	29
Terminology.....	30
3.1 Web Applications	30
3.2 Websites.....	31
3.2.1 Social Network Sites.....	31
3.2.2 Collaborative Web Applications.....	31
3.2.3 E-Commerce Sites.....	31
3.3 Web Services	31
3.3.1 XML-RPC.....	32
3.3.2 SOAP, WSDL and WS-*	32
Study Session Summary	33
Assessment.....	33

Study Session 4 34

Beginning Web Programming with HTML and XHTML	34
Introduction	34
Terminology.....	34
4.1 A Web of Structured Documents	35
4.2 HTML and XHTML.....	35
A very simple web page.....	36
4.3 Tags and Elements	37
4.3.1 Tags	37
4.3.2 Elements: Parent and Child	38
4.4 Parts of a Web Page	38
4.5 Attributes.....	38
4.5.1 Attribute Groups.....	39
4.6 More on Elements	39

4.7 Comments.....	47
Study Session Summary.....	48
Assessment.....	49

Study Session 5	50
------------------------	-----------

Advance HTML (Links, Images, Tables and Forms).....	50
Introduction.....	50
Terminology.....	50
5.1 Creating Links with the <a> Element.....	51
5.1.1 Creating a Source Anchor with the href Attribute.....	51
5.1.2 Advanced E-mail Links.....	53
5.2 Images and Objects.....	54
Types of Image Formats.....	54
5.2.1 Bitmap Images.....	54
5.2.1.1 Bitmap graphics formats.....	55
Keeping File Sizes Small.....	55
5.2.2 Vector Images.....	56
Adding Other Objects with the <object> Element.....	56
5.3 Using Images as Links.....	57
5.3.1 Image Maps.....	57
5.3.1.1 Server-Side Image Maps.....	57
5.3.2 Client-Side Image Maps.....	58
5.4 Tables.....	58
5.4.1 Basic Table Elements and Attributes.....	60
5.4.1.1 The <table> Element.....	60
5.4.1.2 The dir Attribute.....	61
5.4.1.3 The frame Attribute (deprecated).....	61
5.4.1.4 The summary Attribute.....	61
5.4.1.5 The width Attribute (deprecated).....	61
5.4.1.6 The <tr> Element Contains Table Rows.....	61
5.4.1.7. The axis Attribute.....	61
5.4.1.8 The bgcolor Attribute (deprecated).....	62
5.4.1.9 The char Attribute.....	62
5.4.1.10 The charoff Attribute.....	62
5.4.1.11 The colspan Attribute.....	62
5.4.1.12 The headers Attribute.....	62
5.4.1.13 The height Attribute (deprecated).....	62
5.4.1.14 The nowrap Attribute (deprecated).....	62
5.4.1.15 The rowspan Attribute.....	63
5.4.1.16 The scope Attribute.....	63
5.5 Forms.....	63
5.5.1 The action Attribute.....	64
5.5.2 The method Attribute.....	64
5.6 Form Controls.....	66
5.6.1 Text Inputs.....	67
5.6.2 Buttons.....	67
5.6.3 Checkboxes.....	67
5.6.4 Radio Buttons.....	68
5.6.4 Select Boxes.....	68

Study Session Summary	69
Assessment.....	70

Study Session 6 **70**

Cascading Style Sheet (CSS).....	70
Introduction	71
Terminology.....	71
6.1 Origin of CSS	71
6.2 Where Can You Add CSS Rules	73
6.2.1 Advantages of External CSS Style Sheets.....	74
6.2.2 Inheritance	75
6.3 Some CSS Properties.....	75
6.3.1 Basic Example	76
Study Session Summary	80
Assessment.....	81

Study Session 7 **82**

Web Programming in JavaScript (Client side).....	82
Introduction	82
Terminology.....	82
7.1 JavaScript.....	83
7.2 Comments in JavaScript.....	84
7.2.1 The <noscript> Element	85
7.2.2 Data types	85
7.3 Variables.....	85
7.3.1 Assigning a Value to a Variable.....	86
7.3.2 Lifetime of a Variable	86
7.4 Operators.....	87
7.4.1 Arithmetic Operators.....	87
7.4.2 Assignment Operators.....	87
7.4.3 Comparison Operators.....	88
7.4.3 Logical or Boolean Operators.....	88
7.4.4 String Operator.....	89
7.5 Keywords.....	89
7.6 Function and Function Call	90
7.6.1 The Return Statement	91
7.7 Practical Tips for Writing Scripts	91
7.7.1 Online Script	91
7.7.2 Reusable Functions.....	92
7.7.3 Using External JavaScript Files	92
7.7.4 Place Scripts in a Scripts Folder	92
7.8 Form Validation	92
7.8.1 Checking Text Fields	93
7.8.2 Required Text Fields	96
7.8.3 Preventing a Form Submission Until a Checkbox Has Been Selected	98
7.8.4 Testing Characters Using Test and Regular Expressions	99
7.9 When Not To Use JavaScript.....	100
7.9.1 Drop-Down Navigation Menus	100
7.9.2 Hiding Your E-mail Address.....	100

7.9.3 Quick Jump Select Boxes	101
7.9.4 Anything the User Requires from Your Site	101
Study Session Summary	101
Assessment	102

Study Session 8 103

Web Programming in PHP (Server side)	103
Introduction	103
Terminology	103
8.1 PHP: Hypertext Preprocessor	104
8.2 Escape to PHP	104
8.3 Commenting PHP Code	105
8.4 PHP is whitespace insensitive	106
8.5 PHP is case sensitive	107
8.6 Statements are Expressions Terminated by Semicolons	107
8.7 Variables	108
8.8 Data types	108
8.9 Constant	109
8.10 Operator Types	110
8.11 Operators Categories	114
8.11.1 Precedence of PHP Operators	114
8.12 PHP Code	115
8.13 Functions	116
8.14 PHP with Forms	116
8.15 POST and GET	118
8.16 PHP with E-MAIL	119
8.17 Error Reporting	121
Study Session Summary	121
Assessment	122

Study Session 9 123

Concurrency Programming for the Web	123
Introduction	123
Terminology	123
9.1 Concurrency	124
9.2 Reduce Latency	125
9.2.1 Hide latency	125
9.2.2 Increase throughput	125
9.3 Multithreading	126
9.3.1 Method 1: Thread creation by implementing Runnable Interface	126
9.3.2 Method 2: Thread creation by extending Thread class	127
Study Session Summary	127
Assessment	128

Study Session 10 129

Website Maintenance	129
Expected duration: 1 week or 2 contact hour	129
Introduction	129
Terminology	129

10.1 Website Maintenance Team.....	129
10.2 Website Scale.....	130
10.3 Website Size.....	130
10.4 Website Complexity.....	131
10.4.1 Basic Website.....	131
10.4.2 Dynamic Website.....	131
10.4.3 Transactional Website.....	131
10.5 Website Activity.....	131
10.6 Regular Website Maintenance Tasks.....	131
10.6.1 Backing up Website.....	132
10.6.2 Monitor Website Outages.....	132
10.6.3 Check Domain Registration Information.....	132
10.6.4 Test Website Speed.....	132
10.6.5 Link Check.....	132
10.6.6 Software Updates.....	132
10.6.7 Analyze Stats.....	133
10.6.8 Traffic Stats.....	133
10.6.9 Reputation Management.....	133
10.7 Development Server or Live Server.....	133
Study Session Summary.....	134
Assessment.....	134

About this course manual

Introduction to Web Programming CSC 293 has been produced by University of Ibadan Distance Learning Centre. All course manuals produced by University of Ibadan Distance Learning Centre are structured in the same way, as outlined below.

How this course manual is structured

The course overview

The course overview gives you a general introduction to the course. Information contained in the course overview will help you determine:

- If the course is suitable for you.
- What you will already need to know.
- What you can expect from the course.
- How much time you will need to invest to complete the course.

The overview also provides guidance on:

- Study skills.
- Where to get help.
- Course assignments and assessments.
- Margin icons.

We strongly recommend that you read the overview *carefully* before starting your study.

The course content

The course is broken down into Study Sessions. Each Study Session comprises:

- An introduction to the Study Session content.
- Study Session outcomes.
- Core content of the Study Session with a variety of learning activities.
- A Study Session summary.
- Assignments and/or assessments, as applicable.
- Bibliography

Your comments

After completing Introduction to Web Programming we would appreciate it if you would take a few moments to give us your feedback on any aspect of this course. Your feedback might include comments on:

- Course content and structure.
- Course reading materials and resources.
- Course assignments.
- Course assessments.
- Course duration.
- Course support (assigned tutors, technical help, etc.)

Your constructive feedback will help us to improve and enhance this course.

Getting around this course manual

Margin icons

While working through this course manual you will notice the frequent use of margin icons. These icons serve to “signpost” a particular piece of text, a new task or change in activity; they have been included to help you to find your way around this course manual.

A complete icon set is shown below. We suggest that you familiarize yourself with the icons and their meaning before starting your study.

 Activity	 Assessment	 Assignment	 Case study
 Discussion	 Group Activity	 Help	 Outcomes
 Note	 Reflection	 Reading	 Study skills
 Summary	 Terminology	 Time	 Tip

Study Session 1

Understanding the World Wide Web

Expected duration: 1 week or 2 contact hour

Introduction

The Internet is a collection of computers around the world connected to each other via a high speed series of networks. The World Wide Web – or Web – consists of a vast assortment of files and documents that are stored on these computers and written in some form of HyperText Markup Language (HTML) that tells browsers how to display the information. The computers that store the files are called servers because they can serve requests from many users at the same time. Users access these HTML files and documents via applications called browsers. In this study session we will discuss the web publishing process and also learn how to evaluate a website.

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 1.1 *explain* the Web publishing process.
- 1.2 *organize* a sample Web site.
- 1.3 *learn* how to evaluate a Web site.

Terminology

Web browser	A web browser (commonly referred to as a browser) is a software application for retrieving, presenting, and traversing information resources on the World Wide Web.
Web Pages	A hypertext document connected to the World Wide Web.
URI	Uniform Resource Identifier (URI) is a string of characters used to identify a resource. Such identification enables interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols.
HTML	Hypertext Markup Language, a standardized system for tagging text files to achieve font, colour, graphic, and

	hyperlink effects on World Wide Web pages.
Server	A computer or computer program which manages access to a centralized resource or service in a network.

1.1 Web Browsers

Web browser

A web browser (commonly referred to as a browser) is a software application for retrieving, presenting, and traversing information resources on the World Wide Web.

Web Pages

hypertext document connected to the World Wide Web.

HTML

HyperText Markup Language

URI

Uniform Resource Identifier (URI) is a string of characters used to identify a resource. Such identification enables interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols.

A Web browser is a program that displays Web pages and other documents on the Web. Unfortunately, different browsers may interpret the HTML of Web pages somewhat differently, and thus, when you create Web pages remember that they may appear different when viewed in various browsers. When a Web page is opened in a browser, the browser reads and interprets the HTML file and formats the Web page for display. If there are references to external files, such as images or multimedia, these files are downloaded from the server and displayed in the browser window. It is important to note that HTML files are text files that only contain references to the external files – you do not “embed” these files into the Web page.



Figure 1.1: Loading of a web page.

1.2 Web Usability

Web site

A location connected to the Internet that maintains one or more web pages.

For a Web site to be usable, it must be convenient and practical for its intended audience. The content, images, navigation, and placement of these elements need to match what the visitor is expecting. Visitors can easily become frustrated and quickly go to another Web site.

1.2.1 Designing Web Pages

We really don't read Web pages – we scan them. When was the last time you read everything on a Web page? Most visitors scan a Web page, looking for specific words or phrases. When they find an item that matches, they try to click that object to get more information. If it isn't what they want, the visitors simply click the back button and look for

something else. The concept of scanning Web pages is similar to how we “read” a newspaper – quickly scanning titles, reading a few lines here and there.

Here are five important things you can do to make sure your visitors see and understand as much as possible about your Web pages:

1. Create a clear visual hierarchy on each page.
2. Take advantage of conventions.
3. Break pages up into clearly defined areas.
4. Make it obvious what’s clickable.
5. Minimize noise.

1.3 Web site Organization

1.3.1 File Naming Conventions

When creating a Web site (or a Web page), there are a few rules for creating filenames. These rules not only apply to HTML files, but to any file or document that is part of your Web site.

1. Use lower-case letters in your file names. You may use upper-case letters, but do so sparingly. Uploading files with capitals into WebCT can create problems. Some older browsers do not locate files that are not exactly specified.
2. Only use numbers and letters in your file names. File names must begin with a letter (not a number). Special characters, except those noted below, should not be used – including #, & and comma. Do not use any spaces within a filename.
3. Representing spaces within a filename: You may use the underscore (‘_’) character or the dash (‘-’) character to represent a space in a filename.
4. File extensions: Use .htm or .html as the file extension when you name your HTML files. Be consistent with the convention you use.

1.4 Site Structure

Every Web site that you build or inherit should have a consistent and simple organization – called a site structure. A site is a collection of HTML files, documents and images contained in a single master folder (the root folder). Within this root folder you can save your documents and subfolders organized in a manner that makes sense to you, as well as to others in your department that may need to edit the information.

We recommend that the structure of your Web site include:

1. A root folder that contains the Web site.
2. A Web page entitled index.htm (or index.html) that resides within the root folder to represent the default homepage for the Web site.
3. An images folder that contains the graphics, illustrations, images and photographs used in your Web pages.

-
4. Additional folders for organizing your content.

1.5 Web Editors

Web Editors

Web page editors are used to write HTML code directly.

Web editors are software programs that allow you to create and edit Web pages in a visual editor or by using a built-in HTML editor. The visual editor allows you to edit and create Web pages without knowing HTML. Some examples are:

1.5.1 Adobe Dreamweaver

Dreamweaver is a popular Web editor and is the editor of choice for many novice and professional Web designers. It is available for free from your college or department Information Technology Consultant, for University-owned computers only.

1.5.2 Notepad++

This is an amazingly powerful source code editor with a vast number of features. Syntax highlighting makes it immediately easier to read and understand your code, for instance. Code folding allows you to collapse some areas while you focus on others. Auto-completion helps you enter code more quickly (and accurately). There's also a powerful search tool, easy document navigation, bookmarking, macro support, and more, all of which is presented in a highly configurable, easy-to-use interface. Go grab a copy immediately.

1.5.3 Bluefish

This is a programmer's editor which also includes plenty of web-related tools and options. This starts with the usual editing tricks: syntax highlighting (ASP.NET, CSS, HTML, JavaScript, PHP and more are supported), code folding, powerful find and search and replace tools, auto-completion, and more.

1.6 Evaluating Your Web site

This sounds easy, but even many veteran Web designers forget to properly evaluate their Web sites. You can have a colleague or friend help with the evaluation process. Determining which criteria to use in your evaluation can be a cumbersome task. Fortunately, there are many free sites on the Web that contain a list of criteria on which to review your Web site. Don't forget to evaluate the sites you link to from your Web pages. Here are a couple of excellent resources that can assist you in evaluating your Web site:

- Checklist for rating Web sites - <http://www.cyberbee.com/design.pdf>

- Criteria for evaluating Web pages (good for reviewing resources linked from your Web pages)
<http://www.library.cornell.edu/olinuris/ref/webcrit.html>

1.6.1 Designing Your Page to Maximize Rankings

Search engines that use programs to automatically index sites are using increasingly sophisticated rules to determine who gets the highest ranking (top) results in a web page. Following are some points to consider when designing your pages to help ensure that your site gets the highest ranking it can:

- The titles of your pages are among the most important words in your site and are one of the most important things indexed. So avoid using titles that just contain words such as “Home Page” and instead go for descriptive titles such as “Wrox Press — Computer Programming Book Publishers.”
- Then on specific pages the title could change to something like “XHTML Programming Books, learn to code and build web sites.” If the words the user types into the search engine are found in your title, the engine will consider your site more relevant. But don’t make the title longer than one sentence or the program will realize you are trying to fool it and count this against you.
- Most search engines look through the text content of a page and will index that, too. The first words tend to be considered the most relevant. So you should try to strategically place the keywords for your site in the text near the start of the page as well as in the title. You can also expand on that list of keywords here.
- If the keywords a user searches on appear in the page with more frequency than other words, then they are considered to be more relevant. However, do not make them appear too frequently — again, the search engine will count this against you.
- If your site uses images instead of text, the site can index only your alt text; so try to make sure any information conveyed with images is also conveyed in text.
- If you try to fool the search engines by repeating keywords in text that is the same colour as the background (so that the repetitive text is invisible to your users), then the search engines can penalize you for this.
- Using keywords that are not related to the subject matter or the content of the site can count against you.
- The more sites that link to yours the better. Some search engines will give you higher priority if you are linked to by lots of other sites. But note that they will also consider which site is linking to you. The site should be relevant to your business — a search engine would not consider a used car dealers linking to a pet shop as being a relevant link.
- The more users who click on links to your site when it comes up in the search engine, the better your rating should be. While things such as the title, keywords in the text, <meta> tags, and

the number of links may help you appear nearer the top of the search engines, if nobody clicks on the links to visit your site, your ranking will soon fall.

It can take a long time to build up your search engine rankings, but constant attention will help you get better and better.

1.7 Moving Your Files to a Web Server

In order for your audience to see the Web pages you create or edit, you need to copy your completed HTML files, documents and images to a Web server account – such as your faculty Web account or a departmental Web account.

1.7.1 Uniform Resource Identifiers (URIs)

URIs, are strings that are used to reference resources. In terms of distributed systems, a URI has three distinct roles—naming, addressing, and identifying resources. We will focus on URIs identifying resources in the WWW, although they can be used for other abstract or physical entities as well. According to the specification, a URI consists of five parts: scheme, authority, path, query and fragment. However, only scheme and path are mandatory, the other parts are optional. Scheme is declaring the type of the URI and thus determines the meaning of the other parts of the URI. If used, authority points to the responsible authority of the referenced resource. In case of http as scheme, this part becomes mandatory and contains the host of the web server hosting the resource. It can optionally contain a port number (80 is implied for http) and authentication data (deprecated). The mandatory path section is used to address the resource within the scope of the scheme (and authority). It is often structured hierarchically. The optional query part provides non-hierarchical data as part of the resource identifier. Fragments can be used to point to a certain part within the resource.

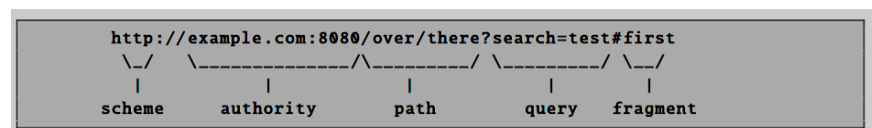


Figure 1.2: Sample URI

It identifies a web resource (scheme is http), that is hosted on the example.com (on port 8080). The resource path is /over/there and the query component contains the key/value pair search test. Furthermore, the first fragment of the resource is referenced.

1.8 What a URL Is Made Up of

URL

URL is an acronym for Uniform Resource Locator

A **URL** is made up of several parts, each of which offers information to the web browser to help find the page you are after. It is easier to learn the parts of a URL if you look at the most common ones first. If you

and is a reference (an address) to a resource on the Internet.

Scheme

The scheme consists of a sequence of characters beginning with a letter and followed by any combination of letters, digits, plus (+), period (.), or hyphen (-).

Host Address

look at the example URL in Figure 1.3, there are three key parts: the scheme, the host address, and the file path. The following sections discuss each of these in turn.

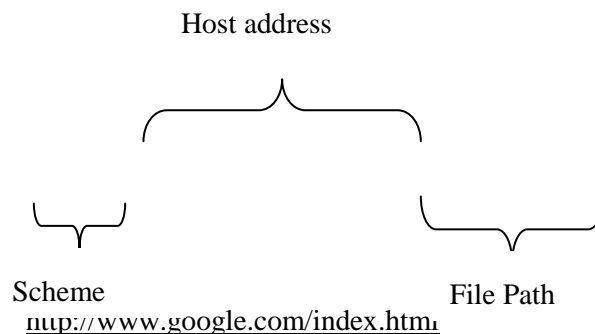


Figure 1.3: Sample URL

1.8.1 The Scheme

The **scheme** identifies the type of URL you are linking to and therefore how the resource should be retrieved. For example, most web pages use something called the Hypertext Transfer Protocol (HTTP) to pass information to you, which is why most web pages start with `http://`, but you might have noticed other prefixes when doing banking online or downloading large files.

The following table lists the most common schemes.

Scheme	Description
<code>http://</code>	Hypertext Transfer Protocol (HTTP) is used to request pages from web servers and send them back from web servers to browsers.
<code>https://</code>	Secure Hypertext Transfer Protocol (HTTPS) encrypts the data sent between the browser and the web server using a digital certificate.
<code>ftp://</code>	File Transfer Protocol is another method for transferring files on the Web. While HTTP is a lot more popular for viewing web sites because of its integration with browsers, FTP is still commonly used to transfer large files across the Web and to upload source files to your web server.
<code>file://</code>	Used to indicate that a file is on the local hard disk or a shared directory on a LAN.

Figure 1.4: List of Schemes

1.8.2 The Host Address

The **host address** is the address where a web site can be found. It can either be an IP address (four sets of numbers between 0 and 255, for example, 192.0.110.255) or more commonly the domain name for a site such as `www.wrox.com`.

All computers connected to the Internet can be found using an IP address; however, domain names are far easier to remember than IP addresses, so domain names are more commonly used. However, behind the scenes, all domain names are converted into the IP address for the computer(s) that hold the web site by consulting a domain name server (DNS), which

contains a directory of domain names and the IP address of the computer that runs that web site.

Note that “www” is not actually part of the domain name although it is often used in the host address — it has nothing to do with the HTTP protocol used.

1.8.3 The File path

The **filepath** always begins with a forward slash character, and may consist of one or more directory names (remember, a directory is just another name for a folder on the web server); each directory name is separated by forward slash characters and the filepath may end with a filename at the end. Here, `Overview.html` is the filename:

`/books/newReleases/BeginningWebDevelopment/Overview.html`

If a filename is not given, the web server will usually do one of three things (depending upon how it is configured):

- Return a default file (for web sites written in HTML this is often `index.html` or `default.html`)
- Offer a list of files in that directory
- Show a message saying that the page cannot be found or that you cannot browse the files in a folder

1.8.4 Other Parts of the URL

A URL may, less commonly, contain a number of other parts.

Credentials are a way of specifying a username and password for a password-protected part of a site.

The credentials come before the host address, and are separated from the host address by an @ sign.

Note how the username is separated from the password by a colon. The following URL shows the username `administrator` and the password `letmein`:

<http://administrator:letmein@www.wrox.com/administration/index.html>

1.8.4.1 Ports

Ports are like the doors to a web server. A web server often has several server programs running on the same machine, and each program communicates using a different port. For example, `http://` and `https://` by default use different ports (standard `http://` usually uses port 80 and `https://` usually uses port 443).

You will rarely have to specify a port, but if you do, it comes after the domain name and is separated from it with a colon. For example, if you wanted to specify that a web server was running on port 8080 you could use the following address:

`http://www.wrox.com:8080/index.html`

1.8.4.2 Fragment Identifiers

Fragment identifiers can be used after a filename to indicate a specific part of the page that a browser should go to immediately. These are often used in long pages when you want to allow a user to get to a specific part of a page easily without having to scroll through the whole page to find that point.

The fragment identifier is separated from the filename by a pound or hash sign:

<http://www.wrox.com/newTitles/index.html#HTML>

1.9 Absolute and Relative URLs

As you have already seen, a URL is used to locate a resource on the Internet. Each web page and image—in fact every file on the Internet—has a unique URL, the address that can be used to find that particular file. No two files on the Internet share the same URL.

If you want to access a particular page of a web site, you type the URL for that page into the address bar in your browser. For example, to get the page about film on the fictional news site you met earlier in the chapter, you might type in the URL:

<http://www.exampleNewsSite.com/Entertainment/Film/index.html>

An absolute URL like this one contains everything you need to uniquely identify a particular file on the Internet.

As you can see, absolute URLs can quickly get quite long, and every page of a web site can contain many links. So it's about time you learned the shorthand for URLs that point to files within your web site: relative URLs.

A relative URL indicates where the resource is in relation to the current page. For example, imagine you are looking at the index page for the entertainment section of the following fictional news site:

<http://www.exampleNewsSite.com/Entertainment/index.html>

Then you want to add a link to the index pages for each of the subsections: Film, TV, Arts, and Music.

Rather than including the full URL for each page, you can use a relative URL. For example:

Film/index.html

TV/index.html

Arts/index.html

Music/index.html

As I am sure you agree, this is a lot quicker than having to write out the following:

<http://www.exampleNewsSite.com/Entertainment/Film/index.html>

<http://www.exampleNewsSite.com/Entertainment/TV/index.html>

`http://www.exampleNewsSite.com/Entertainment/Arts/index.html`

`http://www.exampleNewsSite.com/Entertainment/Music/index.html`

Another key benefit to using relative URLs within your site is that it means you can change your domain name or copy a subsection of one site to a new site without having to change all of the links because each link is relative to other pages within the same site.

Note that relative URLs work only on links within the same directory structure on the same web site; you cannot use them to link to pages on other servers.

1.9.1 Different Types of Relative URLs

1.9.1.1 Same Directory

When you want to link to or include a resource from the same directory, you can just use the name of that file. For example, to link from the home page (`index.html`) to the “contact us” page (`contactUs.html`), you can use the following: `contactUs.html`

Because the file lives in the same folder, you do not need to specify anything else.

1.9.1.2 Subdirectory

The Film, TV, Arts, and Music directories from Figure 2-4 were all subdirectories of the Entertainment directory. If you are writing a page in the Entertainment directory, you can create a link to the index page of the subdirectories like so:

`Film/index.html`

`TV/index.html`

`Arts/index.html`

`Music/index.html`

You include the name of the subdirectory, followed by a forward slash character, and the name of the page you want to link to.

For each additional subdirectory, you just add the name of the directory followed by a forward slash character. So, if you are creating a link from a page in the root folder of the site (such as the site’s main home page), you use a relative URL like these to reach the same pages:

`Entertainment/Film/index.html`

`Entertainment/TV/index.html`

`Entertainment/Arts/index.html`

`Entertainment/Music/index.html`

1.9.1.3 Parent Directory

If you want to create a link from one directory to its parent directory (the directory that it is in), you use the `../` notation of two periods or dots followed by a forward slash character. For example, from a page in the Music directory to a page in the Entertainment directory, your relative URL looks like this:

```
../index.html
```

If you want to link from the Music directory to the root directory, you repeat the notation:

```
../../index.html
```

Each time you repeat the `../` notation, you go up another directory.

1.9.1.4 From the Root

It is also possible to indicate a file relative to the root folder of the site. So, if you wanted to link to the `contactUs.html` page from any page within the site, you use its path preceded by a forward slash. For example, if the Contact Us page is in the root folder, you just need to enter:

```
/contactUs.html
```

Alternatively, you can link to the Music section's index page from anywhere within that site using the following:

```
/Entertainment/Music/index.html
```

The forward slash at the start indicates the root directory, and then the path from there is specified.

1.9.1.5 Default Files

You may have noticed on many web sites that you do not need to actually specify the exact page that you want to view. For example, you might just enter the domain name or the domain name and a directory, such as:

or

```
http://www.exampleNewsSite.com/
```

```
http://www.exampleNewsSite.com/Entertainment/
```

1.10 The Hypertext Transfer Protocol

HTTP

Set of rules for transferring files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web.

The **Hypertext Transfer Protocol** (HTTP) is an application-level protocol that represents the foundation of communication for the WWW on top of TCP/IP. HTTP, is a stateless protocol and complies with a client/server architecture and a request/response communication model. Servers host resources that are identified by URIs and can be accessed by clients. The client issues an HTTP request to the server which in return provides an HTTP response. The communication model limits the possible message patterns to single request/response cycles that are always initiated by the client. Apart from clients and servers, HTTP also describes optional intermediaries, so called proxies. These components

provide additional features such as caching or filtering. Proxies combine features of a client and a server and are thus often transparent for the clients and servers in terms of communication.

HTTP requests and responses have a common structure. Both start with a request line respectively status line. The next part contains a set of header lines that include information about the request respectively response and about the entity.

the entity is an optional body of an HTTP message that contains payload such as a representation of the resource. While the first two parts of an HTTP message are text-based, the entity can be any set of bytes. HTTP request lines contain a request URI and a method. There are different HTTP methods that provide different semantics when applied to a resource. In the subsequent HTTP response, the server informs the client about the outcome of a request by using predefined status codes.

Study Session Summary



Summary

In this Study Session, you learnt that

1. The Internet is a collection of computers around the world connected to each other via a high speed series of networks. The World Wide Web – or Web – consists of a vast assortment of files and documents that are stored on these computers and written in some form of HyperText Markup Language (HTML).
2. A site is a collection of HTML files, documents and images contained in a single master folder (the root folder).
3. Every Web site that you build or inherit should have a consistent and simple organization – called a site structure.
4. A Web site should include: a root folder, an index, an images folder, and additional folders for organizing your content.
5. A Web browser is a program that displays web pages and other documents on the Web.
6. Hypertext Transfer Protocol (HTTP) is an application-level protocol that represents the foundation of communication for the WWW on top of TCP/IP.

Assessment



1. What is the difference between Internet and the web?
2. State five features of a standard website.
3. What do you understand by Uniform Resource Locator (URL)?
4. What is a Web browser?
5. Differentiate between an Absolute and relative URL?

Assignment

6. Differentiate between HTTP and HTML.
7. What are web editors? Give examples.
8. What is a Web browser?

Study Session 2

Web Programming

Expected duration: 1 week or 2 contact hour

Introduction

In the previous study session, we examined the World Wide Web, how to evaluate a website and also the web publishing process. In this study session, we will learn web programming, programming languages used to design and develop a website, client and server side scripting as well as database technology.

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 2.1 *explain* the concept of web programming.
- 2.2 *explain* the term client side scripting and server side scripting.
- 2.3 *analyse* client side scripting and server side scripting.

Terminology

Web Programming	Refers to the writing, markup and coding involved in Web development, which includes Web content, Web client and server scripting and network security.
Client side	Refers to operations that are performed by the client in a client–server relationship in a computer network. Typically, a client is a computer application, such as a web browser, that runs on a user's local computer or workstation and connects to a server as necessary.
Server side	(Commonly referred to as SS) refers to operations that are performed by the server in a client–server relationship in a

	computer network. Typically, a server is a computer program, such as a web server, that runs on a remote server, reachable from a user's local computer or workstation.
--	---

2.1 Web Programming

Web Programming

Refers to the writing, markup and coding involved in Web development, which includes Web content, Web client and server scripting and network security.

Client side

Refers to operations that are performed by the client in a client-server relationship in a computer network. Typically, a client is a computer application, such as a web browser, that runs on a user's local computer or workstation and connects to a server as necessary.

Server side

(Commonly referred to as SS) refers to operations that are performed by the server in a client-server relationship in a computer network. Typically, a server is a computer program, such as a web server, that runs on a remote server, reachable from a user's local computer or workstation.

Web programming can be briefly categorized into client and server coding. The **client side** needs programming related to accessing data from users and providing information. It also needs to ensure there are enough plug-ins to enrich user experience in a graphic user interface, including security measures.

1. To improve user experience and related functionalities on the client side, JavaScript is usually used. It is an excellent client-side platform for designing and implementing Web applications.
2. HTML5 and CSS3 support most of the client-side functionality provided by other application frameworks.

The **server side** needs programming mostly related to data retrieval, security and performance. Some of the tools used here include ASP, Lotus Notes, PHP, Java and MySQL. There are certain tools/platforms that aid in both client- and server-side programming. Some examples of these are Opa and Tersus.

2.2 Client-side Versus Server-side Scripting

Web development is all about communication. In this case, communication between two (2) parties, over the HTTP protocol:

- The Server - This party is responsible for serving pages.
- The Client - This party requests pages from the Server, and displays them to the user. In most cases, the client is a web browser.
 - The User - The user uses the Client in order to surf the web, fill in forms, watch videos online, etc.

Each side's programming, refers to code which runs at the specific machine, the server's or the client's.

Server-side programming is writing code that runs on the server, using languages supported by the server (such as Java, PHP, C#; it is possible to write code that executes on the server-side in JavaScript). Client-side programming is writing code that will run on the client, and is done in languages that can be executed by the browser, such as JavaScript.

Basic Example

1. The User opens his web browser (the Client).

-
2. The User browses to <http://google.com>.
 3. The Client (on the behalf of the User), sends a request to <http://google.com> (the Server), for their home page.
 4. The Server then acknowledges the request, and replies the client with some meta-data (called headers), followed by the page's source.
 5. The Client then receives the page's source, and renders it into a human viewable website.
 6. The User types Stack Overflow into the search bar, and presses Enter
 7. The Client submits that data to the Server.
 8. The Server processes that data, and replies with a page matching the search results.

The Client, once again, renders that page for the User to view.

2.2.1 Client Side Scripting

Client side development is done almost exclusively in JavaScript. This is, of course, in addition to basic HTML and CSS code. The reason JavaScript is called a client side language is because it runs scripts on your computer after you've loaded a web page. Client-side script can be used to enhance the functionality and user experience. For example, it can be used to provide simple mouse over image effects, animations, and form field validations.

2.2.2 Client-side Environment

The client-side environment used to run scripts is usually a browser. The processing takes place on the end users computer. The source code is transferred from the web server to the user's computer over the internet and run directly in the browser.

The scripting language needs to be enabled on the client computer. Sometimes if a user is conscious of security risks they may switch the scripting facility off. When this is the case a message usually pops up to alert the user when script is attempting to run.

Uses

- Make interactive webpages.
- Make stuff happen dynamically on the web page.
- Interact with temporary storage, and local storage (Cookies, localStorage).
- Send requests to the server, and retrieve data from it.
- Provide a remote service for client-side applications, such as software registration, content delivery, or remote multi-player gaming.

Example languages

- JavaScript (primarily)
- HTML*
- CSS*
- Any language running on a client device that interacts with a remote service is a client-side language.

*HTML and CSS aren't really "programming languages" per-se. They are markup syntax by which the Client renders the page for the User.

2.2.3 Server side Scripting

A server side language runs its scripts before the HTML is loaded, not after.

There are a range of server side languages in use on the web today. PHP is one of the most popular, as well as Ruby on Rails, ASP.NET and many others. They are called server side languages because their scripts are run not on your computer, but on the server which hosts the website and sends down the HTML code.

Server-side scripts can also be used to enhance the functionality of a site. Some examples of features that can be included in a site by using server-side scripts include forums, polls, guest books, and searches.

2.2.3 Server-side Environment

The server-side environment that runs a scripting language is a web server. A user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages. This HTML is then sent to the client browser. It is usually used to provide interactive web sites that interface to databases or other data stores on the server. This is different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript. The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.

Uses

- Process user input.
- Display pages.
- Structure web applications.
- Interact with permanent storage (SQL, files).

Example Languages

- PHP
- ASP.Net in C#, C++, or Visual Basic.
- Nearly any language (C++, C#, Java). These were not designed specifically for the task, but are now often used for application-level web services.

2.3 Deployment and platform

Deployment

Deploying your application means putting it on a Web server so that it can be used either through the Internet or an intranet.

In server-side programming, **deployment** has to happen from outside your code, using some kind of tool (even if it is make install or a git clone), and this deployment is usually manual — or at least, it is expected to happen in a semi-supervised way. The system (meaning the OS) on which you deploy is usually uniform across a number of machines, but it can be heavily customized to your needs. In client-side programming, deployment happens from your server-side code, which serves the clients automatically and without supervision. The underlying system (meaning mainly the browser) can be very different across a much larger number of machines. In order to make deployment feasible at all, standards have to be kept, and there is a much stronger trend to a single language and environment. This is why copying server-side code from one machine to another can take weeks, while client-side code is usually trivial to execute in different machines.

2.4 State and secondary effects

In server-side programming, state is a much bigger concern, meaning how to retrieve and update data at the request of the user with the possibility of conflicts due to concurrency. Even if most of this complexity is offloaded to a database server, it is the server-side code's responsibility to allow the database to keep its guarantees on data integrity by using its interface correctly (e.g. not use a cache for updates that are never seen by the DB), while it is also a goal of the server-side code not to overload the database with work and keep the user waiting for response.

In client-side programming, presenting the results to the user is a much bigger concern, and this implies secondary effects (mostly printing to the screen). This is not to say that there is no state involved (e.g. cookies), only that the main goal of the code is to actually interface with the user, and this cannot happen without secondary effects.

This is why client-side programming usually requires (at some point) looking at the screen with a demo, to check that all colors and layout are right, while server-side programming can happen almost exclusively in a text-oriented environment, where automated tests check that the logic is still doing what it is supposed to do.

Study Session Summary



Summary

In this Study Session, you learnt that

1. The Server - This party is responsible for serving pages.
2. The Client - This party requests pages from the Server, and

displays them to the user. In most cases, the client is a web browser.

3. Web programming can be briefly categorized into client and server coding. The client side needs programming related to accessing data from users and providing information. The server side needs programming mostly related to data retrieval, security and performance.
4. Client side development is done almost exclusively in JavaScript.
5. Example languages of client side scripting JavaScript, HTML, CSS, any language running on a client device.
6. Example languages of client side scripting PHP, ASP.Net in C#, C++, or Visual Basic and nearly any language (C++, C#, Java).

Assessment



Assignment

- 1 What is Web programming?
- 2 Differentiate between client side and server side scripting
- 3 Give examples of client side and server side scripting.
- 4 What is Web programming?
- 5 Differentiate between client side and server side scripting
- 6 Give examples of client side and server side scripting.

Study Session 3

Web Application

Expected duration: 1 week or 2 contact hour

Introduction

In the previous study session, we discussed the concept of web programming, identified the difference between client side and server side scripting and also how to deploy a website. Therefore in this study session, we will learn about web based applications, web services and various examples of website based on their purposes.

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 3.1 *differentiate* between web base application and web services;
- 3.2 *examine* what a website is
- 3.3 *describe* web services

Terminology

Web Application	(Web app) is an application program that is stored on a remote server and delivered over the Internet through a browser interface.
Web Service	(Sometimes called application services) are services (usually including some combination of programming and data, but possibly including human resources as well) that are made available from a business's Web server for Web users or other Web-connected programs.

3.1 Web Applications

Web Application

(Web app) is an application program that is stored on a remote server and delivered over the Internet through a browser interface.

Web applications refer to applications accessed via Web browser over a network and developed using browser-supported languages (e.g., HTML, JavaScript). For execution, Web applications depend on Web browsers and include many familiar applications such as online retail sales, online auctions, and webmail. It is a collection of servlets, html pages, classes, and other resources that can be bundled and run on multiple containers from multiple vendors. One of the main characteristics of a Web application is its relationship to the Servlet Context. Each Web application has one and only one Servlet Context. This relationship is controlled by the servlet container and guarantees that Web applications will not clash when storing objects in the Servlet Context.

Web applications are needed in the area of business-to-business interaction over networks, e.g., for overseas companies that outsource projects to each other. The adoption of a Web applications infrastructure can provide vital processes such as transfer of funds and updates of pricing information.

Because of the complexity of service systems, analysis of each component and subsystem becomes more challenging. In the field of Web engineering, the need exists for methodologies for the development of Web services. Web Services provide tools

3.2 Websites

Social Network

A dedicated website or other application which enables users to communicate with each other by posting information, comments, messages, images, etc.

Web sites have evolved from hyper-referenced, text-based research documents to highly interactive, social and collaborative applications for many different purposes. Web content has become increasingly dynamic and based on content provided by the users. Web technologies such as JavaScript (JS), AJAX, and HTML5 have introduced more interactive user interfaces and boosted this transition. Thanks to powerful APIs provided by modern browsers, web applications are already starting to replace traditional desktop applications and native mobile applications.

Examples

We will now introduce some popular types for web sites that are interesting in terms of scalability and concurrency.

3.2.1 Social Network Sites

Social networks are sites that transfer social interactions to the web. They often try to reflect real world social relations of its users (e.g. Facebook) or focus on specific topics (e.g. dopplr for travelling). Social network sites motivate their users to interact, for instance via instant messaging. Also, social networks heavily rely on user generated content and context-specific data, such as geo-tagged content. User content and actions are often published into activity streams, providing a “real-time” feed of updates that can be watched live by other users.

3.2.2 Collaborative Web Applications

These web applications allow a group of people to collaborate via the web. Traditional examples are wiki systems, where users can collectively edit versions of text documents. More recent collaborative applications incorporate some real-time aspects. For example, Etherpad is a web-based word processor that supports multiple users working on the same document concurrently.

3.2.3 E-Commerce Sites

E-Commerce sites such as Amazon are traditional commercial sites in the web selling products online. Interestingly enough, many sites have adopted features known from social sites for business purposes. By commenting, rating and tagging products, users can participate on these sites beyond just buying items. User-generated content is then used to cluster product items and compute accurate recommendations. Thus, commercial sites face similar challenges to social sites to some extent.

3.3 Web Services

Web Services

(Sometimes called application services) are

Web services provide access to application services using HTTP. Thus, web services often resemble traditional mechanisms for distributed computing such as Remote Procedure Call (RPC) or message passing,

services (usually including some combination of programming and data, but possibly including human resources as well) that are made available from a business's Web server for Web users or other Web-connected programs.

though based on web technologies. Opposed to web sites, web services are not targeting direct (human) user access in a first place. Instead, web services enable machine-to-machine communication and provide application features via an interface and structured messages. Several web applications provide both, a web site and a web service interface (API). While the web site is used for browser-based access, the web service can be used for custom applications such as mobile client applications or scripted program-based service interactions.

3.3.1 XML-RPC

XML-RPC has been one of the first attempts to transfer traditional RPC-based services to the web. It makes use of HTTP POST requests for dispatching procedure calls and an XML-based serialization format for call parameters and return values. It is important to clarify that XML-RPC is using HTTP as a generic **transport protocol** for RPC calls. It does not take advantage of any HTTP features such as caching, status codes for error handling or header fields for negotiation.

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-Length: 181

<?xml version="1.0"?> <methodCall> <methodName>examples.getStateName</
methodName> <params> <param> <value><i4>41</i4></value> </param> </
params> </methodCall>

HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?> <methodResponse> <params> <param> <value><string>South
Dakota</string></value> </param> </params> </methodResponse>
```

Figure 3.1: Example XML-RPC call

3.3.2 SOAP, WSDL and WS-*

The stack of SOAP, WSDL, UDDI and a myriad of additional extensions (WS-*) forms a more comprehensive approach for machine-to-machine communication, mostly based on web technologies. It is widely used and particularly popular in enterprise environments.

As previously mentioned, SOAP is a successor of XML-RPC. It specifies the format, call semantics and exchange patterns of XML-encoded messages between parties. Various extension specifications, often labeled as WS-*, address additional features of SOAP-based web services such as security or orchestration. The Web Services Description Language (WSDL) is another important specification for this kind of web services. It provides XML-based, machine-readable service descriptions, comparable to interface definition languages of traditional RPC protocols. Universal Description, Discovery and Integration (UDDI) was originally

a third component providing registry functions for web services, but it has almost entirely lost its significance in practice.

Although the SOAP/WSDL stack is generally known as web service stack, it dismisses parts of the original idea of the web. For instance, the stack uses HTTP as a pure communication protocol for exchanging messages that can be replaced by other protocols. Similarly to XML-RPC, this web service stack does not use HTTP as an application-level protocol.

Study Session Summary



Summary

In this Study Session, you learnt that

1. Web applications depend on Web browsers and include many familiar applications. It is a collection of servlets, html pages, classes, and other resources that can be bundled and run on multiple containers from multiple vendors. Web applications infrastructure can provide vital processes such as transfer of funds and updates of pricing information.
2. Some popular types for web sites that are interesting in terms of scalability and concurrency are Social Network Sites e.g. Facebook, Collaborative Web Applications e.g. Etherpad, E-Commerce Sites e.g. ebay.
3. XML-RPC has been one of the first attempts to transfer traditional RPC-based services to the web. It makes use of HTTP POST requests for dispatching procedure calls and an XML-based serialization format for call parameters and return values.

Assessment



Assignment

1. What is a web base application?
2. What is a web service application?
3. What is a website?
4. Give 3 examples each of the 2 above.

Study Session 4

Beginning Web Programming with HTML and XHTML

Expected duration: 1 week or 2 contact hour

Introduction

In the previous study session, we distinguished between web applications and web services. Therefore in this study session, we will examine critically HTML and XHTML as the basics for building websites.

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 4.1 *examine* the difference between tags, elements, and attributes
- 4.2 *analyze* how a web page uses markup to describe how the page should be structured
- 4.3 *use* different HTML elements to format a web page.
- 4.4 *differentiate* between HTML and XHTML

Terminology

HTML Tags	
	An HTML code that defines every structure on an HTML page, including the placement of text and images and hypertext links. HTML tags begin with the less-than (<) character and end with greater-than (>). These symbols

	are also called "angle brackets."
HTML Elements	An individual component of an HTML document or web page, once this has been parsed into the Document Object Model. HTML is composed of a tree of HTML elements and other nodes, such as text nodes. Each element can have HTML attributes specified.
HTML Attributes	An HTML attribute is a modifier of an HTML element type. An attribute either modifies the default functionality of an element type or provides functionality to certain element types unable to function correctly without them. In HTML syntax, an attribute is added to an HTML start tag.

4.1 A Web of Structured Documents

The Web is like a sea of documents all linked together; these documents bear a strong similarity to the documents that you meet in everyday life. For example, I have a form sitting on my desk (which I really must mail) from an insurance company. This form contains fields for me to write my name, address, and the amount of coverage I want, and boxes I have to check to indicate the number of rooms in the house and what type of lock I have on my front door. Indeed, there are lots of forms on the Web, from a simple search box that asks what you are looking for to the registration forms you are required to go through before you can place an online order for books or CDs. Consider another example: Say I'm catching a train to see a friend, so I check the schedule to see what time the trains go that way. The main part of the schedule is a table telling me what times trains arrive and when they depart from different stations. In the same way that a lot of documents have headings and paragraphs, a lot of other documents use tables; from the stocks and shares pages in the financial supplement of your paper to the TV listings at the back, you come across tables of information every day—and these are often recreated on the Web.

As you can see, there are many parallels between the structure of printed documents you come across every day and pages you see on the Web. So you will hardly be surprised to learn that when it comes to writing web pages, your code tells the web browser the structure of the information you want to display— what text to put in a heading, or in a paragraph, or in a table, and so on—so that the browser can present it properly to the user. In order to tell a web browser the structure of a document—how to make a heading, a paragraph, a table, and so on—you need to learn HTML and XHTML.

4.2 HTML and XHTML

XHTML, or Extensible Hypertext Markup Language, and its predecessor HTML, are the most widely used languages on the Web. As its name suggests, XHTML is a markup language, the key purpose of this kind of markup is to provide a structure that makes the document easier to

understand. When marking up documents for the Web, you are performing a very similar process, except you do it by adding things called tags to the text. With XHTML the key thing to remember is that you are adding the tags to indicate the structure of the document, which part of the document is a heading, which parts are paragraphs, what belongs in a table, and so on. Browsers such as Internet Explorer, Firefox, and Safari will use this markup to help present the text in a familiar fashion, similar to that of a word processor (headings are bigger than the main text, there is space between each paragraph, lists of bullet points have a circle in front of them). However the way these are presented is up to the browser; the XHTML specification does not say which font should be used or what size that font should be. While earlier versions of HTML allowed you to control the presentation of a document—things like which typefaces and colors a document should use—XHTML markup is not supposed to be used to style the document; that is the job of CSS.

A very simple web page

Simply use a text editor such as Notepad on Windows or TextEdit on a Mac, and save your files with an .html file extension.

```
<html>
  <head>
    <title>Popular Websites: Google</title>
  </head>
  <body>
    <h1>About Google</h1>
    <p>Google is best known for its search engine, although
      Google now offers a number of other services.</p>
    <p>Google's mission is to organize the world's
      information and make it universally accessible and
      useful.</p>
    <p>Its founders Larry Page and Sergey Brin started
      Google at Stanford University.</p>
  </body>
</html>
```

There are several sets of angle brackets with words or letters between them, such as

`<html>`, `<head>`, `</title>`, and `</body>`. These angle brackets and the words inside them are known as tags, and these are the markup we have been talking about.

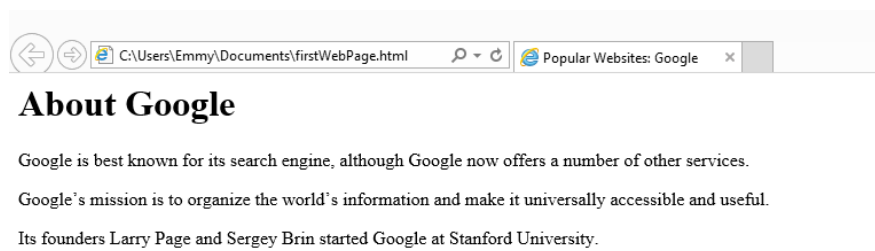


Figure 4.1: First HTML Page

As you can see, this document contains the heading “About Google” and a paragraph of text to introduce the company. Note also that it says “Popular Websites: Google” in the top-left of the browser window; this is known as the title of the page.

4.3 Tags and Elements

HTML Tag

An HTML code that defines every structure on an HTML page, including the placement of text and images and hypertext links. HTML tags begin with the less-than (<) character and end with greater-than (>). These symbols are also called "angle brackets."

HTML Elements

An individual component of an HTML document or web page, once this has been parsed into the Document Object Model. HTML is composed of a tree of HTML elements and other nodes, such as text nodes. Each element can have HTML attributes specified.

4.3.1 Tags

If you look at the first and last lines of the code for the last example, you will see pairs of angle brackets containing the letters <html>. The two brackets and all of the characters between them are known as a tag, and there are lots of **tags** in the example. All the tags in this example come in pairs; there are opening tags and closing tags. The closing tag is always slightly different from the opening tag in that it has a forward slash after the first angled bracket </html>. A pair of tags and the content these include are known as an element.

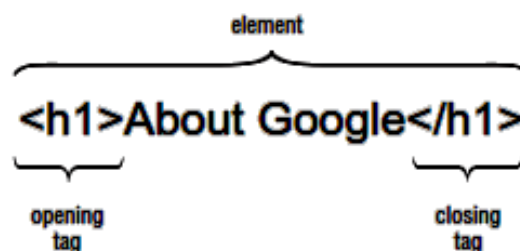


Figure 4.2: Opening and Closing tags

The opening tag says “This is the beginning of a heading” and the closing tag says “This is the end of a heading.” Like most of the tags in XHTML, the text inside the angled brackets explains the purpose of the tag—here h1 indicates that it is a level 1 heading (or top-level heading). As you will see shortly, there are also tags for subheadings (<h2>, <h3>, <h4>, <h5>, and <h6>). Without the markup, the words “About Google” in the middle of the tags would just be another bit of text; it would not be clear that they formed the heading.

4.3.2 Elements: Parent and Child

Tags are the angle brackets and the letters and numbers between them, whereas **elements** are tags and anything between the opening and closing tags.

You will often find that terms from a family tree are used to describe the relationships between elements. For example, an element that contains another element is known as the parent, while the element that is between the parent element's opening and closing tags is called a child of that element. So, the `<title>` element is a child of the `<head>` element, the `<head>` element is the parent of the `<title>` element, and so on. Furthermore, the `<title>` element can be thought of as a grandchild of the `<html>` element.

4.4 Parts of a Web Page

There are two main parts to the page:

- The `<head>` element: Often referred to as the head of the page, this contains information about the page (this is not the main content of the page). It is information such as a title and a description of the page, or keywords that search engines can use to index the page. It consists of the opening `<head>` tag, the closing `</head>` tag, and everything in between. Inside the `<head>` element of the first example page, you can see a `<title>` element:

```
<head>  
    <title>Popular Websites: Google</title>  
</head>
```
- The `<body>` element: Often referred to as the body of the page, this contains the information you actually see in the main browser window. It consists of the opening `<body>` tag, closing `</body>` tag, and everything in between. The real content of your page is held in the `<body>` element, which is what you want users to read, and is shown in the main browser window.

4.5 Attributes

HTML Attributes

An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag. All attributes are made up of two parts: a name and a value

Attributes are used to say something about the element that carries them, and they always appear on the opening tag of the element that carries them. All attributes are made up of two parts: a name and a value:

- The name is the property of the element that you want to set. In this example, the `<a>` element carries an attribute whose name is `href`, which you can use to indicate where the link should take you.
- The value is what you want the value of the property to be. In this example, the value was the URL that the link should take you to,

so the value of the href attribute is <http://www.google.com>. The value of the attribute should always be put in double quotation marks, and it is separated from the name by the equal sign. If you wanted the link to open in a new window, you could add a target attribute to the opening <a> tag as well, and give it a value of _blank:

```
<a href="http://www.Google.com" target="_blank">
```

This illustrates that elements can carry several attributes, although an element should never have two attributes of the same name.

4.5.1 Attribute Groups

As you have seen, attributes live on the opening tag of an element and provide extra information about the element that carries them. All attributes consist of a name and a value; the name reflects a property of the element

The attribute is describing, and the value is a value for that property.

For example, the xml:lang attribute describes the language used within that element; a value such as EN-US would indicate that the language used inside the element is U.S. English.

There are three groups of attributes that many of the XHTML elements can carry (as you have already seen, the <html>, <head>, <title>, and <body> elements share some of these attributes).

The three attribute groups are:

- Core attributes: The class, id, and title attributes
- Internationalization attributes: The dir, lang, and xml:lang attributes
- UI events: Attributes associated with events onclick, ondoubleclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, and onkeyup. Together, the core attributes and the internationalization attributes are known as the universal attributes.

4.6 More on Elements

Element Category	Type	Description	Example
Formatting	Hn where n is any number between 1 & 6	XHTML offers six levels of headings, which use the elements <h1>, <h2>, <h3>, <h4>, <h5>, and <h6>. While browsers can display	<h1>Basic Text Formatting</h1>

		<p>headings differently, they tend to display the <h1> element as the largest of the six and <h6> as the smallest, CSS can be used to override the size and style of any of the elements.</p>	
	<p>	<p>Creating Paragraphs</p>	<p><p>Here is a paragraph of text.</p></p> <p><p>Here is a second paragraph of text.</p></p> <p><p>Here is a third paragraph of text.</p></p>
	 	<p>Creating Line Breaks.</p> <p>Whenever you use the
 element, anything following it starts on the next line. The
 element</p> <p>is an example of an empty element, where you do not need opening and closing tags, because there is nothing to go in between them.</p>	
	<pre>	<p>Creating Preformatted Text.</p> <p>Any text between the opening <pre> tag and the closing </pre> tag will preserve the formatting of the source document. Two of the most common uses of the <pre> element are to display tabular data without the use of a table (in which case you must use the monospaced font or columns will not align correctly) and to represent computer source code. For example, the following shows some JavaScript inside a <pre> element</p>	<pre><pre> function testFunction(strText){ alert (strText) } </pre></pre>
Presentationa		Anything that appears in a 	The following word

<p>1 Elements (these elements affect only the presentation of a document. The full list is bold, italic, monospaced, underlined, strikethrough, teletype, larger, smaller, superscripted, and subscripted text.)</p>		<p>element is displayed in bold.</p> <p>This <code></code> element has the same effect as the <code></code> element, which you will meet later, and is used to indicate that its contents have strong emphasis.</p>	<p>would be <code>bold</code></p>
	<code><i></code>	<p>The content of an <code><i></code> element is displayed in italicized text, like the word italic.</p> <p>The <code><i></code> element has the same effect as the <code></code> element, which you will meet later, and which is used to indicate that its contents have emphasis.</p>	<p>The following word would be <code><i>italized</i></code></p>
	<code><u></code>	<p>The content of a <code><u></code> element is underlined with a simple line</p>	<p>The following word would be <code><u>underlined</u></code></p>
	<code><s></code> or <code><strike></code>	<p>The content of an <code><s></code> or <code><strike></code> element is displayed with a strikethrough, which is a thin line through the text (- <code><s></code> is just the abbreviated form of <code><strike></code>).</p>	<p>The following word would have a <code><s>strikethrough</s></code>.</p>
	<code><sup></code>	<p>The <code><sup></code> element is especially helpful in adding exponential values to equations, and adding the st, nd, rd, and th suffixes to numbers such as dates. However, in some browsers, you should be aware that it can create a taller gap between the line with the superscript text and the line above it.</p>	<p>Written on the 31<code><sup>st</sup></code> February.</p>
	<code><sub></code>	<p>The content of a <code><sub></code> element is written in subscript; the font size used is the same as the characters surrounding it, but is displayed half a character's height beneath the other characters.</p>	<p>The EPR paradox<code><sub>2</sub></code> was devised by Einstein, Podolsky, and Rosen.</p>
	<code><big></code>	<p>The content of the <code><big></code> element is displayed one font size larger than the rest of the text surrounding it. If the font is already the largest size, it has no</p>	<p>The following word should be <code><big>bigger</big></code> than those around it.</p>

		effect. You can nest several <code><big></code> elements inside one another, and the content of each will get one size larger for each element.	
	<code><small></code>	The content of the <code><small></code> element is displayed one font size smaller than the rest of the text surrounding it. If the font is already the smallest, it has no effect. You can nest several <code><small></code> elements inside one another, and the content of each gets one size smaller for each element.	The following word should be <code><small>smaller</small></code> than those around it.
	<code><hr /></code>	The <code><hr /></code> element creates a horizontal rule across the page. It is an empty element, rather like the <code>
</code> element. This is frequently used to separate distinct sections of a page where a new heading is not appropriate.	<code><hr /></code>
Phrase Elements (designed to describe their content)	<code></code>	For emphasis	<code><p>You <code>must</code> remember to close elements in XHTML.</p></code>
	<code></code>	The <code></code> element is intended to show strong emphasis for its content—stronger emphasis than the <code></code> element.	<code><p><code>Always</code> look at burning magnesium through protective colored glass as it <code>can cause blindness</code>.</p></code>
	<code><blockquote e></code> ,	When you want to quote a passage from another source, you should use the <code><blockquote></code> element. Use the <code>cite</code> attribute on the <code><blockquote></code> element to indicate the source of the quote.	<code><p>The following description of XHTML is taken from the W3C Web site:</p> <code><blockquote></code> XHTML 1.0 is the W3C's first Recommendation for XHTML, <code></blockquote></code> following on from earlier work on HTML 4.01,</code>

			HTML 4.0, HTML 3.2 and HTML 2.0. </blockquote>
	<cite>	If you are quoting a text, you can indicate the source by placing it between an opening <cite> tag and closing </cite> tag. As you would expect in a print publication, the content of the <cite> element is rendered in italicized text by default.	This chapter is taken from <cite>Beginning Web Development</cite>.
	<q>	The <q> element is intended to be used when you want to add a quote within a sentence rather than as an indented block on its own	<p>As Dylan Thomas said, <q>Somebody's boring me. I think it's me</q>.</p>
	<abbr>	for abbreviations	I have a friend called <abbr title="Beverly">Bev</abbr>.
	<acronym>	The <acronym> element allows you to indicate that the text between an opening <acronym> and closing </acronym> tags is an acronym.	This chapter covers marking up text in <acronym title="Extensible Markup Language">XHTML</acronym>.
	<dfn>	The <dfn> element allows you to specify that you are introducing a special term. Its use is similar to the words that are in italics in the midst of paragraphs in this book when new key concepts are introduced.	This book teaches you how mark up your documents for the Web using <dfn>XHTML</dfn>.
	<code>	for computer code and information	<p><code><h1>Th is is a primary heading</h1></code></p>
	<kbd>	For text typed on the keyboard	<p>Type in the

			following: <code><kbd>This is the kbd element</kbd>.</p></code>
	<code><samp></code>	The <code><samp></code> element indicates sample output from a program, script, or the like. Again, it is mainly used when documenting programming concepts.	<code><p>If everything worked you should see the result <samp>Test completed OK</samp>.</p></code>
	<code><var></code>	The <code><var></code> element is another of the elements added to help programmers. It is usually used in conjunction with the <code><pre></code> and <code><code></code> elements to indicate that the content of that element is a variable that can be supplied by a user	<code><p><code>document.write("<var>user-name</var>")</code></p></code>
	<code><address></code>	for addresses	<code><address>Wrox Press, 10475 Crosspoint Blvd, Indianapolis, IN 46256</address></code>
List	There are three types of lists in XHTML: <input type="checkbox"/> Unordered lists, which are like lists of bullet points <input type="checkbox"/>		<code></code> <code><ul compact="compact"></code> <code>Item one</code> <code>Item two</code> <code>Item three</code> <code></code>

	<p><input type="checkbox"/> Ordered lists, which use a sequence of numbers or letters instead of bullet points</p>		<pre> Point number one Point number two Point number three </pre>
	<p>Definition lists, which allow you to specify a term and its definition.</p> <p>The definition list is a special kind of list for providing terms followed by a short text definition or description for them. Definition lists are contained inside the <dl> element. The <dl> element then contains alternating <dt> and <dd> elements. The content of</p>		<pre><dl> <dt>Unordered List</dt> <dd>A list of bullet points.</dd> <dt>Ordered List</dt> <dd>An ordered list of points, such as a numbered set of steps.</dd> <dt>Definition List</dt> <dd>A list of terms and definitions.</dd> </dl></pre>

	the <code><dt></code> element is the term you will be defining.		
Editing	<code><ins></code>	The <code><ins></code> element for when you want to add text	<pre> <h1>How to Spot a Wrox Book</h1> <p>Wrox-spotting is a popular pastime in bookshops. Programmers like to find the distinctive blue<ins>red </ins> spines because they know that Wrox books are written by 1000 monkeys <ins>Programmers</ins> for Programmers.</p> <ins><p>Both readers and authors, however, have reservations about the use of photos on the covers.</p></ins> </pre>
	<code></code>	The <code></code> element for when you want to delete some text	<pre> <h1>How to Spot a Wrox Book</h1> <p>Wrox-spotting is a popular pastime in bookshops. Programmers like to find the distinctive blue<ins>red </ins> spines because they know that Wrox books are written by 1000 monkeys <ins>Programmers</ins> for Programmers.</p> <ins><p>Both readers and authors, however, have reservations about the use of photos on the covers.</p></ins> </pre>

Grouping	<div>	The <div> element is used to group block-level elements:	<pre><div class="footnotes"> <h2>Footnotes</h2> <p>1 The World Wide Web was invented by Tim Berners-Lee</p> <p>2 The W3C is the World Wide Web Consortium who maintain many Web standards</p> </div></pre>
		The element, on the other hand, can be used to group inline elements only.	<pre><div class="footnotes"> <h2>Footnotes</h2> <p>1 The World Wide Web was invented by Tim Berners Lee</p> <p>2 The W3C is the World Wide Web Consortium who maintain many Web standards</p> </div></pre>

4.7 Comments

You can put comments between any tags in your XHTML documents. Comments use the following syntax:

```
<!-- comment goes here -->
```

Anything after <!-- until the closing --> will not be displayed. It can still be seen in the source code for the document, but it is not shown onscreen. It is good practice to comment your code, especially in complex documents, to indicate sections of a document, and any other notes to anyone looking at the code. Comments help you and others understand your code.

Question

.

Feedback

.

Study Session Summary



Summary

In this Study Session, you learnt that

1. HTML and XHTML are needed to explain the structure of any web pages. They're used to indicate what text should be considered a heading, where paragraphs start and end, and what images should appear in the document, and to specify links between different pages.
2. Tags are the angle brackets and the letters and numbers between them, whereas elements are tags and anything between the opening and closing tags.
3. An element that contains another element is known as the parent, while the element that is between the parent element's opening and closing tags is called a child of that element.
4. There are two main parts to the page: `<head>` element and the `<body>` element.
5. Attributes are used to say something about the element that carries them, and they always appear on the opening tag of the element that carries them.
6. The three attribute groups are: Core attributes (the class, id, and title attributes); the Internationalization attributes (the dir, lang, and xml:lang attributes), UI events (Attributes associated with events: onclick, ondoubleclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, and onkeyup).
7. These elements consist of an opening tag, a closing tag, and some content between the opening and closing tags. In order to alter some properties of elements, the opening tag may carry attributes, and attributes are always written as name value pairs.
8. By default, most browsers display the contents of the `<h1>`, `<h2>`, and `<h3>` elements larger than the default size of text in the document. The content of the `<h4>` element would be the same size as the default text, and the content of the `<h5>` and `<h6>` elements would be smaller.
9. The six levels of headings: `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and

<h6>

10. Paragraphs <p>, preformatted sections <pre>, line breaks
, and addresses <address>
11. Presentational elements , <i>, <u>, <s>, <tt>, <sup>, <sub>, <strike>, <big>, <small>, and <hr />
12. It is good practice to comment your code, especially in complex documents, to indicate sections of a document, and any other notes to anyone looking at the code. Comments help you and others understand your code.

Assessment



Assignment

1. What are tags, elements and attribute?
2. Mention the categories of elements
3. Give three examples of each category in 2 above.

Study Session 5

Advance HTML (Links, Images, Tables and Forms)

Expected duration: 1 week or 2 contact hour

Introduction

In the previous study session, we discussed how to use HTML elements to format a webpage, difference between Tags, Elements and Attributes. In this study session, we will examine some advance HTML elements and the role they play in building a good web application.

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 5.1 *explain* the meaning of links.
- 5.2 *use* images as links.
- 5.3 *state* the usefulness of Image maps.
- 5.4 *explain* the types of Image format.
- 5.5 What client side and server side Image maps are

Terminology

Vector Image	The use of polygons to represent images in computer graphics. Vector graphics are based on vectors, which lead through locations called control points or nodes. Each of these points has a definite position on the x and y axes of the work plane and determines the direction of the path; further, each path may be assigned a stroke color, shape, curve, thickness, and fill
Bitmap Image	A bit map (often spelled "bitmap") defines a display space and the color for each pixel or "bit" in the display space. A Graphics Interchange Format and a JPEG are examples of graphic image file types that contain bit maps.
HREF tag	(Hypertext REFerence) The HTML code used to create a link to another page. The HREF is an attribute of the anchor tag, which is also used to identify sections within a document.

5.1 Creating Links with the <a> Element

HREF

(Hypertext REFerence) The HTML code used to create a link to another page. The HREF is an attribute of the anchor tag, which is also used to identify sections within a document.

All hypertext links on the Web take you from one part of the Web to another. You have already seen links that take you from one page to another (and this section covers them in more depth). You will also meet links that take you to a specific part of a page (either a specific part of the same page or specific part of a different page).

Like all journeys, these have a starting point known as the source, and a finishing point known as the destination, which are both called anchors.

Each link that you see on a page that you can click is actually a source anchor, and each source anchor is created using the <a> element.

5.1.1 Creating a Source Anchor with the href Attribute

The source anchor is what most people think of when talking about links on the Web — whether the link contains text or an image. It is something you can click and then expect to be taken somewhere else.

As you have already seen, any text that forms part of the link that a user can click is contained between the opening <a> tag and closing tag, and the URL to which the user should be taken is specified as the value of the **href** attribute.

For example, when you click the words Wrox Press website (which you can see are inside the <a> element) the link takes you to <http://www.wrox.com/>:

Why not visit the [Wrox Press website](http://www.wrox.com/) to find out about some of our other books? Whereas the following link on the home page of the fictional news site would take you to the main Film page (note how this link uses a relative URL):

You can see more films in the [film](Entertainment/Film/index.html)

section. You need to specify a destination anchor only if you want to link to a specific part of a page.

Creating a Destination Anchor Using the name and id Attributes (linking to a specific part of a page)

If you have a long web page, you might want to link to a specific part of that page. You will usually want to do this when the page does not fit in the browser window, and the user might otherwise have to scroll to find the relevant part of the page.

The destination anchor allows the page author to mark specific points in a page that a source link can point to.

Common examples of linking to a specific part of a page that you might have seen used on web pages include:

- “Back to top” links at the bottom of long pages

-
- A list of contents for a page that takes the user to the relevant section
 - Links to footnotes or definitions

You create a destination anchor using the `<a>` element again, but when it acts as a destination anchor it must carry an `id` attribute (and if you are creating pages that might be viewed by very early browsers, such as IE 3 and Netscape 3, a `name` attribute as well) because the `id` attribute was only introduced in HTML 4.

By way of an example, imagine that you have a long page with a main heading and several subheadings.

The whole page does not fit on the screen at once, forcing the user to scroll, so you want to add links to each of the main headings at the start of the document.

Before you can create links to each section of the page (using the source anchors), you have to add the destination anchors. Here you can see the subheadings of the page, each containing an `<a>` element with the `id` attribute whose value uniquely identifies that section:

```
<h1>Linking and Navigation</h1>
<h2><a id="URL">URLs</a></h2>
<h2><a id="SourceAnchors">Source Anchors</a></h2>
<h2><a id="DestinationAnchors">Destination Anchors</a></h2>
<h2><a id="Examples">Examples</a></h2>
```

With destination anchors in place, it's now possible to add source anchors to link to these sections, like so:

```
<p>This page covers the following topics:
<ul>
<li><a href="#URL">URLs</a></li>
<li><a href="#SourceAnchors">Source Anchors</a></li>
<li><a href="#DestinationAnchors">Destination Anchors</a></li>
<li><a href="#Examples">Examples</a></li>
</ul>
</p>
```

The value of the `href` attribute in the source anchors is the value of the `id` attribute preceded by a pound or hash sign (`#`).

If someone wanted to link to a specific part of this page from a different web site, he or she would add the full URL for the page, followed by the pound or hash sign and then the value of the `id` attribute, like so:

`http://www.example.com/HTML/links.html#SourceAnchors.`

5.1.2 Advanced E-mail Links

You can make a link open up the user's default e-mail editor, and address an e-mail to you —or any other e-mail address you give —automatically. This is done like so:

```
<a href="mailto:info@example.org">info@example.org</a>
```

You can also specify some other parts of the message, too, such as the subject, body, and people that it should be cc'd or bcc'd to.

To add a subject to an e-mail, you follow the e-mail address with a question mark to separate the extra values from the e-mail address. Then you use the name/value pairs to specify the additional properties of the mail you want to control. The name and the value are separated by an equal sign.

For example, to set the subject to be Enquiry, you would add the subject property name and what you wanted to be the subject, like so:

```
<a href="mailto:info@example.org?subject=Enquiry">
```

You can specify more than one property by separating the name/value pairs with an ampersand. Here you can see that the subject and a cc address have been added in:

```
<a href="mailto:info@example.org?subject=XHTML&cc=sales@example.org"></a>
```

The table that follows includes a full list of properties you can add.

Property	Purpose
subject	Adds a subject line to the e-mail; you can add this to encourage the user to use a subject line that makes it easier to recognize where the mail has come from.
body	Adds a message into the body of the e-mail, although you should be aware that users would be able to alter this message
cc	Sends a carbon copy of the mail to the cc'd address; the value must be a valid e-mail address. If you want to provide multiple addresses you simply repeat the property, separating it from the previous one with an ampersand.
bcc	Secretly sends a carbon copy of the mail to the bcc'd address without any recipient seeing any other recipients; the value must be a valid e-mail address. If you want to provide multiple addresses, you simply repeat the property, separating it from the previous one with an ampersand.

If you want to add a space between any of the words in the subject line, you should add %20 between the words instead of the space. If you want to take the body part of the message onto a new line you should add %0D%0A (where 0 is a zero, not a capital O).

It is common practice to add only the e-mail address in e-mail links. If you want to add subject lines or message bodies you are better off creating an e-mail form.

5.2 Images and Objects

Bitmap Image

A bit map (often spelled "bitmap") defines a display space and the color for each pixel or "bit" in the display space. A Graphics Interchange Format and a JPEG are examples of graphic image file types that contain bit maps.

Vector Image

The use of polygons to represent images in computer graphics. Vector graphics are based on vectors, which lead through locations called control points or nodes. Each of these points has a definite position on the x and y axes of the work plane and determines the direction of the path; further, each path may be assigned a stroke color, shape, curve, thickness, and fill

Images and graphics can really bring your site to life. However, it is important to choose the right format for your images and save them correctly as this will help make your site faster and result in happier visitors.

Types of Image Formats

Graphics are created for computers in two main ways:

- Bitmapped graphics divide a picture into a grid of pixels and specify the color of each pixel, much as a computer tells a screen the color of each pixel. Broadly speaking, bitmaps are ideal for photographs and complicated gradations of shade and color. There are several different Bitmap formats; common ones include JPEG, GIF, TIFF, PNG, and the rather confusingly named bitmap or BMP.
- **Vector graphics** break the image into lines and shapes (like a wireframe drawing), and store the lines as coordinates. They then fill the spaces between the lines with color. Vector graphics are commonly used for line art, illustration, and animation. They often feature large areas of flat color (as opposed to textures, shades of colors, and photographic styles).

In the early days, bitmaps were the main image format for the Web, although more recently some formats such as Flash and SVG are making use of vector graphics.

5.2.1 Bitmap Images

Most static images on the Web are **bitmapped images**. The image is divided into a grid of pixels. If you look very closely at your computer screen you may be able to make out the pixels that make up the screen. If you look at Figure 5.1, you can see an example of a bitmap image with one section that has been modified so that you can see how pixels make up the image.

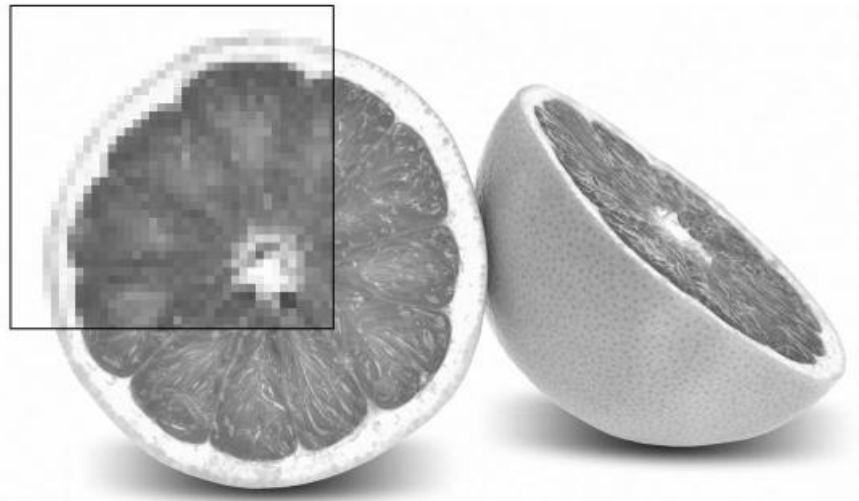


Figure 5.1: Illustrating Bitmap image

The number of pixels in every square inch of the screen is known as the resolution of the image. Images on the Web can show a maximum of 72 pixels per inch; images used in print are usually higher resolution and are often supplied to printers at 300 dots per inch (note how onscreen we refer to pixels per inch, while in print we call them dots per inch). The more pixels or dots per inch an image contains, the larger the size of the file will be. As a result, any images that you use on the Web, you save at a resolution of 72 dots per inch. If you saved it any larger, this would create unnecessarily large files that would take longer to download.

5.2.1.1 Bitmap graphics formats

Browsers tend to support three common bitmap graphics formats, and most graphics programs will save images in these formats:

- GIF: Graphics Interchange Format (pronounced either “gif” or “jif”)
- JPEG: Joint Photographic Experts Group Format (pronounced “jay peg”)
- PNG: Portable Network Graphics (pronounced “ping” or “pee en gee”)

Keeping File Sizes Small

You will usually want to save the images for your site in the format that best compresses the image and therefore results in a smaller file size. This will not only make your pages quicker to load, but can also save you on the charges made for hosting your site.

Usually one or another format will be the obvious choice for you. The rule of thumb is:

- Use JPEGs for photo-realistic pictures with a lot of detail, or subtle shade differences you want to preserve.
- Use GIFs for images with flat color (rather than textured colors), and hard edges, such as diagrams, text, or logos.

You can also consider using PNGs if you do not need the advanced features such as transparency, or if you know the majority of your visitors will be using more recently released browsers.

5.2.2 Vector Images

Illustration and animation software tends to use vector formats to save images, and the most popular vector graphics format on the Web is Flash. Vector formats store information in terms of coordinates between which lines are drawn, and then inside the lines a colored fill can be specified. Because vector formats are based on the coordinates that mark points on lines, it is very easy for vector formats to scale to different sizes simply by increasing or decreasing the gap between each point the coordinates are plotted against.

Browsers and XHTML do not, by default, support any vector graphics formats, although the main browsers now ship with the Flash Player that is required to view Flash files. As a result, Flash is currently the most popular way of deploying vector graphics and animations on the Web. While the Flash Player is free for download, and the browsers feature it, you should be aware that Adobe charges for the software to create Flash files and that learning to use the software is an entirely new skill.

Adding Other Objects with the `<object>` Element

Before the `<object>` element was introduced, a range of elements was used to insert multimedia objects into pages, such as the `<applet>`, `<embed>` and `<bgsound>` elements, but these elements have been deprecated. The `<object>` element was initially introduced by Microsoft to support its Active X technology; however, it was soon used to embed all kinds of object in web pages. To embed an object into a page, you need to specify:

- The location of the code used to display or play the object (sometimes referred to as the implementation of the object)
- The actual data to be rendered (for example a movie, an audio file, a program)
- Any additional values the object needs at runtime

The first two are added using the `<object>` element, while additional values are provided in the `<param>` element, which can be a child of the `<object>` element. While the `<object>` element can contain a child `<param>` element, any other content of the `<object>` element should be displayed only if the browser cannot render the object:

```
<object>Your browser does not appear to support the format used in this film clip,
```

```
for more details please look <a href=" ../help/video.htm">here</a>
```

```
</object>
```


5.3 Using Images as Links

It's easy to turn an image into a link; rather than putting text between the opening `<a>` tag and the closing `` tag, as you saw in the last chapter, you can place an image inside these tags. Images are often used to create graphical buttons or links to other pages,

```
<a href="../index.html" title="Click here to return to the home page">  
</a>
```

Note the use of the deprecated `border` attribute. When you use an image inside an `<a>` element, the image will gain a border in IE for Windows.

5.3.1 Image Maps

Image maps allow you to specify several links that correspond to different areas of one single image, so that when users click different parts of the image they get taken to different pages. There are two types of image maps:

- Server-side image maps
- Client-side image maps

The difference between the two lies in where the code that decides which link you should be taken to is executed. With client-side image maps, the browser indicates which page you should be taken to based upon where the user clicks, whereas with server-side image maps the browser sends the server the coordinates of where the user clicked, and these are processed by a script file on the server that determines which page the user should be sent to. Image maps are particularly helpful when the image needs to be divided up in irregular shapes, such as maps. However, if the image can be divided up in a grid. These hotspots should not be too small; otherwise, users might have difficulty in selecting the correct area they want. If this happens, they will soon get frustrated and leave your site. Image maps can also be difficult for people with motor control difficulties to navigate. Thus, if for any reason you use image maps as the main method of navigation for your site you should offer text links at the bottom of the page (and indicate this in the alt text).

5.3.1.1 Server-Side Image Maps

With server-side images, the `` element (inside an `<a>` element) carries a special `ismap` attribute, which tells the browser to send the server x, y coordinates representing where the user's mouse was when he or she clicked the image map. Then a script on the server is used to determine which page the user should be sent to based on the coordinates fed to it.

For example, look at the following link, where the `` element carries the `ismap` attribute with a value of `ismap` (this is an attribute that did not require a value in HTML; however, in XHTML all attributes must have a value, and therefore its own name is used as a value in XHTML to make the attribute valid):

```
<a href="../location/map.aspx"></a>
```

Now, if the user clicks the image 50 pixels to the right of the top-left corner of the image and 75 pixels down from the that same corner, the browser will send this information with the URL like so: `http://www.example.org/location/map.aspx?50,75`

You can see the coordinates appended at the end of the URL that is specified in the `<a>` element.

5.3.2 Client-Side Image Maps

Because server-side image maps rely on server technology, an alternative that worked on browsers was introduced and client-side image maps were born. Client-side image maps use code within the XHTML page to indicate which parts of the image should link to which pages. Because the code that divides up the sections of the image is on the browser, it is possible for the browser to offer extra information to users, either by showing them a URL in the status bar or as a tooltip when the mouse is hovered over the image.

There are two methods of creating a client-side image map: using the `<map>` and `<area>` elements inside an `` element, and, more recently, using the `<map>` element inside the `<object>` element.

The thing about a server-side image map is that there needs to be a script, map file, or application on the server that can process the coordinates and know which page the user should then be sent to. The implementation of image maps will vary depending on what kind of server you are running on.

5.4 Tables

Tables are commonly used to display all manner of data, such as timetables, financial reports, and sports results. So when you want to display information in rows and columns, you need to use the markup to create a table. In order to work with tables, you need to start thinking in grids. Here you can see a grid of rectangles. Each rectangle is known as a cell. A row is made up of a set of cells on the same line from left to right, while a column is made up of a line of cells going from top to bottom.

By now you have understood that the names of elements in XHTML tend to refer to the type of markup they contain. So you will hardly be surprised to know that you create a table in XHTML using the `<table>` element.

Inside the `<table>` element, the table is written out row by row. A row is contained inside a `<tr>` element — which stands for table row. And each cell is then written inside the row element using a `<td>` element — which stands for table data.

Column 1 Row 1	Column 2 Row 1	Column 3 Row 1	Column 4 Row 1
Column 1 Row 2	Column 2 Row 2	Column 3 Row 2	Column 4 Row 2
Column 1 Row 3	Column 2 Row 3	Column 3 Row 3	Column 4 Row 3
Column 1 Row 4	Column 2 Row 4	Column 3 Row 4	Column 4 Row 4
Column 1 Row 5	Column 2 Row 5	Column 3 Row 5	Column 4 Row 5

Figure 5.1: An Example of a Table

The following is an example of a very basic table

```
<table border="1">
  <tr>
    <td>Row 1, Column 1</td>
    <td>Row 1, Column 2</td>
  </tr>
  <tr>
    <td>Row 2, Column 1</td>
    <td>Row 2, Column 2</td>
  </tr>
</table>
```

All tables will follow this basic structure, although there are additional elements and attributes that allow you to control the presentation of tables. If a row or column should contain a heading, a `<th>` element is used in place of the table data or `<td>` element. By default, most browsers render the content of a `<th>` element in bold text.

Here you can see a slightly more complex example of a table, which includes headings

```
<table border="1">
  <tr>
    <th></th>
    <th>Outgoings ({$})</th>
    <th>Receipts ({$})</th>
    <th>Profit ({$})</th>
  </tr>
</table>
```

```

</tr>
<tr>
  <th>Quarter 1 (Jan-Mar)</th>
  <td>11200.00</td>
  <td>21800.00</td>
  <td><b>10600.00</b></td>
</tr>
<tr>
  <th>Quarter 2 (Apr-Jun)</th>
  <td>11700.00</td>
  <td>22500.00</td>
  <td><b>10800.00</b></td>
</tr>
<tr>
  <th>Quarter 3 (Jul - Sep)</th>
  <td>11650.00</td>
  <td>22100.00</td>
  <td><b>10450.00</b></td>
</tr>
<tr>
  <th>Quarter 4 (Oct - Dec)</th>
  <td>11850.00</td>
  <td>22900.00</td>
  <td><b>11050.00</b></td>
</tr>
</table>

```

5.4.1 Basic Table Elements and Attributes

5.4.1.1 The `<table>` Element

The `<table>` element is the containing element for all tables. It can carry the following attributes:

- All of the universal attributes
- Basic event attributes for scripting

The `<table>` element can carry the following deprecated attributes. Even though they are deprecated, you will still see many of them in use today:

align, bgcolor, border, cellpadding, cellspacing, dir, frame, rules, summary, width

5.4.1.2 The dir Attribute

The dir attribute is supposed to indicate the direction of text that is used in the table. Possible values are ltr for left to right text and rtl for right to left (for languages such as Hebrew and Arabic): dir="rtl"

If you use the dir attribute with a value of rtl on the <table> element, then the cells appear from the right first and each consecutive cell is placed to the left of that one.

5.4.1.3 The frame Attribute (deprecated)

The frame attribute is supposed to control the appearance of the outermost border of the whole table, referred to here as its frame, with greater control than the border attribute. If both the frame and border attributes are used, the frame attribute takes precedence. The syntax is: frame="frameType"

5.4.1.4 The summary Attribute

The summary attribute is supposed to provide a summary of the table's purpose and structure for nonvisual browsers such as speech browsers or Braille browsers.

The value of this attribute is not rendered in IE or Firefox, but you should include it in your pages for accessibility purposes: summary="Table shows the operating profit for the last four quarters. The first column indicates the quarter, the second indicates outgoings, the third indicates receipts, and the fourth indicates profit."

5.4.1.5 The width Attribute (deprecated)

The width attribute is used to specify the width of the table in pixels, or as a percentage of the available space. When the table is not nested inside another element, the available space is the width of the screen; otherwise the available space is the width of the containing element.

width="500" or width="90%"

5.4.1.6 The <tr> Element Contains Table Rows

The <tr> element is used to contain each row in a table. Anything appearing within a <tr> element should appear on the same row. It can carry five attributes, four of which have been deprecated in favor of using CSS.

5.4.1.7. The axis Attribute

The axis attribute allows you to add conceptual categories to cells, and therefore represent n-dimensional data. The value of this attribute would be a comma-separated list of names for each category the cell belonged to.

axis="heavy, old, valuable"

Rather than having a visual formatting effect, this attribute allows you to preserve data, which then maybe used programmatically, such as querying for all cells belonging to a certain category.

5.4.1.8 The bgcolor Attribute (deprecated)

The bgcolor attribute sets the background color for the cell. The value of this attribute should be either a hex code or a color name.

5.4.1.9 The char Attribute

The char attribute specifies a character, the first instance of which should be used to horizontally align the contents of a cell. (See the full description in the “The char Attribute” subsection within the “The <tr> Element Contains Table Rows” section earlier in the chapter.)

5.4.1.10 The charoff Attribute

The charoff attribute specifies the number of offset characters that can be displayed before the character specified as the value of the char attribute.

5.4.1.11 The colspan Attribute

The colspan attribute is used to specify how many columns of the table a cell will span across. The value of the colspan attribute is the number of columns the cell stretches across.

5.4.1.12 The headers Attribute

The headers attribute is used to indicate which headers correspond to that cell. The value of the attribute is a space-separated list of the header cells’ id attribute values:

headers=”income q1”.

The main purpose of this attribute is to support voice browsers. When a table is being read to you it can be hard to keep track of which row and column you are on; therefore the header attribute is used to remind users which row and column the current cell’s data belongs to.

5.4.1.13 The height Attribute (deprecated)

The height attribute allows you to specify the height of a cell in pixels or as a percentage of the available space: height=”20” or height=”10%”

5.4.1.14 The nowrap Attribute (deprecated)

The nowrap attribute is used to stop text from wrapping onto a new line within a cell. You would use nowrap only when the text really would not make sense if it were allowed to wrap onto the next line (for example a line of code that would not work if it were spread across two lines). In HTML it was used without

An attribute value, but that would not be allowed in Transitional XHTML. Rather,

you would use the following: nowrap=”nowrap”

5.4.1.15 The rowspan Attribute

The rowspan attribute specifies the number of rows of the table a cell will span across, the value of the attribute being the number of rows the cell stretches across.

5.4.1.16 The scope Attribute

The scope attribute can be used to indicate which cells the current header provides a label or header information for. It can be used instead of the headers attribute in basic tables, but does not have much support: scope="range"

The <td> and <th> Elements Represent Table Cells Every cell in a table will be represented by either a <td> element for cells containing table data or a <th> element for cells containing table headings.

By default the contents of a <th> element are usually displayed in a bold font, horizontally aligned in the center of the cell. The content of a <td> element, meanwhile, will usually be displayed left-aligned and not in bold (unless otherwise indicated by CSS or another element).

The <td> and <th> elements can both carry the same set of attributes, each of which applies just to that cell. Any effect these attributes have will override settings for the table as a whole or any containing element (such as a row).

In addition to the universal attributes and the basic event attributes, the <td> and <th> elements can also carry the following attributes:

abbr, align, axis, bgcolor, char, charoff, colspan, headers, height, nowrap, rowspan, scope, valign, width.

5.5 Forms

Any form that you create will live inside an element called <form>. Between the opening <form> and closing </form> tags, you will find the form controls (the text input boxes, drop-down boxes, checkboxes, a submit button, and so on). A <form> element can also contain other XHTML markup just like the rest of a page.

Once users have entered information into a form, they usually have to click what is known as a submit button (although the actual text on the button may say something different such as Search, Send, or Proceed — and often pressing the return key on the keyboard has the same effect as clicking this button).

This indicates that the user has filled out the form, and this usually sends the form data to a web server.

Once the data that you have entered arrives at the server, a script or other program usually processes the data and sends a new web page back to you. The returned page will usually respond to a request you have made or acknowledge an action you have taken.

The <form> element carries an attribute called action whose value is the URL of the page on the web server that handles search requests. The

method attribute meanwhile indicates which HTTP method will be used in getting the form data to the server.

The `<form>` element can also contain other markup, such as paragraphs, headings, and so on. A `<form>` element must not, however, contain another `<form>` element.

Providing you keep your `<form>` elements separate from each other (and no one `<form>` element contains another `<form>` element), your page may contain as many forms as you like. For example, you might have a login form, a search form, and a form to subscribe to a newsletter all on the same page. If you do have more than one form on a page, users will be able to send the data from only one form at a time to the server.

Every `<form>` element should carry at least two attributes: `action`, `method`

5.5.1 The action Attribute

The `action` attribute indicates what happens to the data when the form is submitted. Usually the value of the `action` attribute is a page or program on a web server that will receive the information from this form when a user presses the submit button.

For example, if you had a login form consisting of a username and password, the details the user enters may get passed to a page written in ASP.net on the web server called `login.aspx`, in which case the `action` attribute would read as follows:

```
<form action="http://www.example.org/membership/login.aspx">
```

Most browsers will accept only a URL beginning with `http://` as the value of the `action` attribute.

5.5.2 The method Attribute

Form data can be sent to the server in two ways, each corresponding to an HTTP method:

The `get` method, which sends data as part of the URL

- ❖ When you send form data to the server using the HTTP `get` method, the form data is appended to the URL specified in the `action` attribute of the `<form>` element.

The form data is separated from the URL using a question mark. Following the question mark you get the name/value pairs for each form control. Each name/value pair is separated by an ampersand (&).

For example, take the following login form, which you saw when the password form control was introduced:

```
<form action=http://www.example.com/login.aspx  
method="get">
```

```
    Username:
```



```
<input type="text" name="txtUsername" value=""  
size="20" maxlength="20"><br />  
Password:  
<input type="password" name="pwdPassword"  
value="" size="20" maxlength="20">  
<input type="submit" />  
</form>
```

When you click the submit button, your username and password are appended to the URL <http://www.example.com/login.aspx> like so in what is known as the query string:

<http://www.example.com/login.aspx?txtUsername=Bob&pwdPassword=LetMeIn>

Note that, when a browser requests a URL with any spaces or unsafe characters (such as /, \, =, &, and +, which have special meanings in URL), they are replaced with a hex code to represent that character. This is done automatically by the browser, and is known as URL encoding. When the data reaches the server, the server will usually un-encode the special characters automatically.

One of the great advantages of passing form data in a URL is that it can be bookmarked. If you look at searches performed on major search engines such as Google, they tend to use the get method so that the page can be bookmarked.

The get method, however, has some disadvantages. Indeed, when sending sensitive data such as the password shown here, or credit card details, you should not use the get method because the sensitive data becomes part of the URL and is in full view to everyone (and could be bookmarked).

You should not use the HTTP get method when:

- You are updating a data source such as a database or spreadsheet (because someone could make up URLs that would alter your data source).
- You are dealing with sensitive information, such as passwords or credit card details (because the sensitive form data would be visible as part of a URL).
- You have large amounts of data (because older browsers do not allow URLs to exceed more than 1,024 characters — although the recent versions of the main browsers do not have limits).
- Your form contains a file upload control (because uploaded files cannot be passed in the URL).
- Your users might enter non-ASCII characters such as Hebrew or Cyrillic characters. In these circumstances, you should use the HTTP post method.

❖ The post method, which hides data in the HTTP headers

When you send data from a form to the server using the HTTP post method, the form data is sent transparently in what is known as the HTTP headers.

While you do not see these headers, they are sent in clear text and cannot be relied upon to be secure (unless you are sending data under a Secure Sockets Layer, or SSL). If the login form you just saw was sent using the post method, it could look something like this in the HTTP headers:

```
User-agent: MSIE 5.5
Content-Type: application/x-www-form-urlencoded
Content-length: 35
...other headers go here...
txtUserName=Bob&pwdPassword=LetMeIn
```

Note that the last line is the form data, and that it is in exactly the same format as the data after the question mark in the get method — it would also be URL — encoded if it contained spaces or any characters reserved for use in URLs.

There is nothing to stop you using the post method to send form data to a page that also contains a query string. For example, you might have one page to handle users that want to subscribe to or unsubscribe from a newsletter, and you might choose to indicate whether a user wanted to subscribe or unsubscribe in the query string. Meanwhile, you might want to send their actual contact details in a form that uses the post method because you are updating a data source. In this case, you could use the following <form> element:

```
<form action=http://www.example.com/newsletter.asp?action=subscribe
method="post">
```

The only issue with using the HTTP post method is that the information the user entered on the form cannot be bookmarked in the same way it can when it is contained in the URL. So you cannot use it to retrieve a page that was generated using specific form data as you can when you bookmark a page generated by most search engines, but it is good for security reasons.

5.6 Form Controls

This section covers the different types of form controls that you can use to collect data from a visitor to your site. They are:

- Text input controls
- Buttons
- Checkboxes and radio buttons
- Select boxes (sometimes referred to as drop-down menus) and list boxes
- File select boxes
- Hidden controls

5.6.1 Text Inputs

You undoubtedly have come across text input boxes on many web pages. Possibly the most famous text input box is the one right in the middle of the Google home page that allows you to enter what you are searching for.

On a printed form, the equivalent of a text input is a box or line that you are allowed to write a response in or on.

There are actually three types of text input used on forms:

- Single-line text input controls: Used for items that require only one line of user input, such as search boxes or e-mail addresses. They are created using the `<input>` element.
- Password input controls: These are just like the single-line text input, except they mask the characters a user enters so that the characters cannot be seen on the screen. They tend to either show an asterisk or a dot instead of each character the user types, so that someone cannot simply look at the screen to see what a user types in. Password input controls are mainly used for entering passwords on login forms or sensitive details such as credit card numbers. They are also created using the `<input>` element.
- Multi-line text input controls: Used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created with the `<textarea>` element.

5.6.2 Buttons

Buttons are most commonly used to submit a form, although they are sometimes used to clear or reset a form and even to trigger client-side scripts. (For example, on a basic loan calculator form within the page, a button might be used to trigger the script that calculates repayments without sending the data to the server.) You can create a button in three ways:

- Using an `<input>` element with a `type` attribute whose value is `submit`, `reset`, or `button`
- Using an `<input>` element with a `type` attribute whose value is `image`
- Using a `<button>` element

With each different method, the button will appear slightly different.

5.6.3 Checkboxes

Checkboxes are just like the little boxes that you have to check on paper forms. As with light switches, they can be either on or off. When they are checked they are on and the user can simply toggle between on and off positions by clicking the checkbox. Checkboxes can appear individually, with each having its own name, or they can appear as a group of checkboxes that share a control name and allow users to select several values for the same property.

Checkboxes are ideal form controls when you need to allow a user to:

-
- Provide a simple yes or no response with one control (such as accepting terms and conditions or subscribing to an e-mail list)
 - Select several items from a list of possible options (such as when you want a user to indicate all of the skills they have from a given list)

A checkbox is created using the `<input>` element whose type attribute has a value of `checkbox`.

5.6.4 Radio Buttons

Radio buttons are similar to checkboxes in that they can be either on or off, but there are two key differences:

- When you have a group of radio buttons that share the same name, only one of them can be selected. Once one radio button has been selected, if the user clicks another option, the new option is selected and the old one deselected.
- You should not use radio buttons for a single form control where the control indicates on or off because once a lone radio button has been selected it cannot be deselected again (without writing a script to do that).

Therefore, radio buttons are ideal if you want to provide users with a number of options from which they can pick only one. In such situations, an alternative is to use a drop-down select box that allows users to select only one option from several. Your decision between whether to use a select box or a group of radio buttons depends on three things:

- Users expectations: If your form models a paper form where users would be presented with several checkboxes, from which they can pick only one, then you should use a group of radio buttons.
- Seeing all the options: If users would benefit from having all the options in front of them before they pick one, you should use a group of radio buttons.
- Space: If you are concerned about space, a drop-down select box will take up far less space than a set of radio buttons.

5.6.4 Select Boxes

A drop-down select box allows users to select one item from a drop-down menu. Drop-down select boxes can take up far less space than a group of radio buttons.

Drop-down select boxes can also provide an alternative to single-line text input controls where you want to limit the options that a user can enter. For example, you can use a select box to allow users to indicate which country or state they live in (the advantage being that all users from Nigeria would have the same value, rather than potentially having people write Nigeria, N.G., ng, or Naija and then having to deal with different answers for the same country).

A drop-down select box is contained by a `<select>` element, while each individual option within that list is contained within an `<option>` element..

Study Session Summary



Summary

In this Study Session, we discussed that

1. The source anchor is what most people think of when talking about links on the Web — whether the link contains text or an image. It is something you can click and then expect to be taken somewhere else.
2. Each link that you see on a page that you can click is actually a source anchor, and each source anchor is created using the `<a>` element.
3. The destination anchor allows the page author to mark specific points in a page that a source link can point to.
4. Images and graphics can really bring your site to life
5. Graphics are created for computers in two main ways: Bitmapped graphics, Vector graphics.
6. Image maps allow you to specify several links that correspond to different areas of one single image, so that when users click different parts of the image they get taken to different pages.
7. There are two types of image maps: Server-side image maps, Client-side image maps.
8. There are two methods of creating a client-side image map: using the `<map>` and `<area>` elements inside an `` element, and, more recently, using the `<map>` element inside the `<object>` element.
9. The thing about a server-side image map is that there needs to be a script, map file, or application on the server that can process the coordinates and know which page the user should then be sent to.
10. Tables are commonly used to display all manner of data. Each rectangle is known as a cell. A row is made up of a set of cells on the same line from left to right, while a column is made up of a line of cells going from top to bottom
11. Every `<form>` element should carry at least two attributes: action, method. The action attribute indicates what happens to the data when the form is submitted.
12. Form data can be sent to the server in two ways, each corresponding to an HTTP method: The get method, which sends data as part of the URL, and the post method, which hides data in the HTTP headers.
13. There are different types of form controls that you can use to collect data from a visitor to your site. They are: Text input controls, Buttons, Checkboxes and radio buttons, Select boxes (sometimes referred to as drop-down menus) and list boxes, File

select boxes, Hidden controls.

Assessment



Assignment

1. What are links?
2. Mention the different ways of linking pages on the web
3. State the usefulness of Image maps
4. List and explain the types of Image format
5. Differentiate between client side and server side Image maps

Study Session 6

Cascading Style Sheet (CSS)

Expected duration: 1 week or 2 contact hour

Introduction

In the previous study session, we learnt how to use some advance HTML elements to structure and add more life to our website. In this study session, we will learn how to beautify how HTML using Cascading Style Sheet (CSS)

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 6.1 *explain* what makes up a CSS rule
- 6.2 *describe* how to place CSS rules within your document, and how to link to an external CSS document
- 6.3 *evaluate* how properties and values control presentation of different elements within your document
- 6.4 *discuss* how CSS is based on a box model, and how you set different properties for these boxes (such as width and styles of borders)

Terminology

CSS	A cascading style sheet (CSS) is a Web page derived from multiple sources with a defined order of precedence where the definitions of any style element conflict.
CSS Selector	A CSS selector is the part of a CSS rule set that actually selects the content you want to style.

6.1 Origin of CSS

CSS

A cascading style sheet (CSS) is a Web page derived from multiple sources with a defined order of precedence where the definitions of any style element conflict.

CSS Selector

A CSS selector is the part of a CSS rule set that actually selects the content you want to style.

Earlier versions of HTML used elements and attributes in the markup of the web page (just like the ones you have met already in the book) to control how a document should appear. However, the W3C decided quite a while back that the HTML and XHTML languages should no longer contain instructions that indicated how the document appears — rather that CSS should be used to control the appearance of web pages.

The cascading style sheets specification works by allowing you to specify rules that say how the content of elements within your document should appear. In fact, you can set different rules to control the appearance of every element in your page so that your pages start to look a lot more interesting.

CSS works by allowing you to associate rules with the elements that appear in the document. These rules govern how the content of those elements should be rendered. Figure 6-1 shows you an example of a CSS

rule, which as you can see is made up of two parts:

- The selector, which indicates which element or elements the declaration applies to (if it applies to more than one element, you can have a comma-separated list of several elements)
- The declaration, which sets out how the elements should be styled

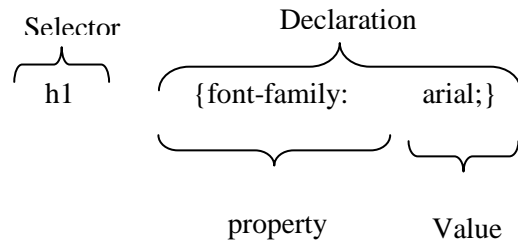


Figure 6.1: Example of CSS Rule

The rule in Figure 6-1 applies to all <h1> elements and indicates that they should appear in the Arial typeface.

The declaration is also split into two parts, separated by a colon:

- A property, which is the property of the selected element(s) that you want to affect, in this case the font-family property.
- A value, which is a specification for this property; in this case it is the Arial typeface.

This is very similar to the way elements can carry attributes in HTML, where the attribute controls a property of the element, and its value would be the setting for that property. With CSS, however, rather than your having to specify the attribute on each instance of the <h1> element, the selector indicates that this one rule applies to all <h1> elements in the document.

Here is an example of a CSS rule that applies to several different elements (in this example, the <h1>, <h2>, and <h3> elements). A comma separates the name of each element that this rule will apply to. The rule also specifies several properties for these elements with each property-value pair separated by a semicolon.

Note how all the properties are kept inside the curly braces:

```
h1, h2, h3 {  
    font-weight:bold;  
    font-family:arial, verdana, sans-serif;  
    color:#000000;  
    background-color:#FFFFFF;}
```

Even if you have never seen a CSS rule before, you should now have a good idea of what this rule is doing.

The content of each heading element named in the selector (<h1>, <h2>,

and `<h3>`) will be written in a bold Arial font (unless the computer does not have Arial installed, in which case it will look for Verdana, failing which its default sans-serif font), and this will be written in black with a white background.

6.2 Where Can You Add CSS Rules

The example that you saw at the beginning of the chapter used a separate style sheet, or external style sheet, to contain the CSS rules. This involved the use of the

`<link />` element in the header of the XHTML document to indicate which style sheet should be used to control the appearance of the document.

CSS rules can also appear in two places inside the XHTML document:

- Inside the `<head>` element, contained with a `<style>` element
- As a value of a style attribute on any element that can carry the style attribute

When the style sheet rules are held inside a `<style>` element in the head of the document, they are referred to as an internal style sheet.

```
<head>
  <title>Internal Style sheet</title>
  <style type="text/css">
    body {
      color:#000000;
      background-color:#ffffff;
      font-family:arial, verdana, sans-serif; }
    h1 {font-size:18pt;}
    p {font-size:12pt;}
  </style>
</head>
```

When style attributes are used on XHTML elements, they are known as inline style rules. For example:

```
<td style="font-family:courier; padding:5px; border-style:solid;
border-width:1px; border-color:#000000;">
```

Here you can see that the properties are added as the value of the style attribute. There is no need for a selector here (because the style is automatically applied to the element that carries the style attribute), and there are no curly braces. You still need to separate each property from its value with a colon and each of the property-value pairs from each other with a semicolon.

6.2.1 Advantages of External CSS Style Sheets

If two or more documents are going to use a style sheet, you should always aim to use an external style sheet (although you may sometimes resort to an internal style sheet to override rules in the external style sheet).

There are several advantages to using external CSS style sheets rather than internal style sheets or inline style rules, including the following:

- The same style sheet can be reused by all of the web pages in your site. This saves you from including the stylistic markup in each individual document.
- Because the style rules are written only once, rather than appearing on every element or in every document, the source documents are smaller. This means that, once the CSS style sheet has been downloaded with the first document that uses it, subsequent documents will be quicker to download (because the browser retains a copy of the CSS style sheet and the rules do not have to be downloaded for every page). This also puts less strain on the server (the computer that sends the web pages to the people viewing the site) because the pages it sends out are smaller.
- You can change the appearance of several pages by altering just the style sheet rather than each individual page; this is particularly helpful if you want to change your company's colors, or the font used for a certain type of element wherever that element appears across the whole site.
- The style sheet can act as a style template to help different authors achieve the same style of document without learning all of the individual style settings.
- Because the source document does not contain the style rules, different style sheets can be attached to the same document. So you can use the same XHTML document with one style sheet when the viewer is on a desktop computer, another style sheet when the user has a handheld device, another style sheet when the page is being printed, another style sheet when the page is being viewed on a TV, and so on. You reuse the same document with different style sheets for different visitors' needs.
- A style sheet can import and use styles from other style sheets, making for modular development and good reuse.
- If you remove the style sheet, you make the site more accessible for those with visual impairments, because you are no longer controlling the fonts and color schemes. It is fair to say, therefore, that whenever you are writing a whole site, you should be using an external style sheet to control the presentation, although as you will see in the next chapter you might use several external style sheets for different aspects of the site.

6.2.2 Inheritance

One of the powerful features of CSS is that many of the properties that have been applied to one element will be inherited by child elements (elements contained within the element that the rules were declared upon). For example, once the font-family property had been declared for the <body> element in the previous example, it applied to all of the elements inside the <body> element (all of the <body> element's child elements).

If a more specific rule comes along, the more specific rule will override any properties associated with the <body> element, or any other containing element. In the preceding example, most of the text was in an Arial typeface, as specified in the rule associated with the <body> element. There were a few table cells that used a Courier typeface. The table cells that were different had a class attribute whose value was code:

```
<td class="code">font-size</td>
```

Here you can see the rule associated with these elements:

```
td.code {  
    font-family:courier, courier-new, serif;  
    font-weight:bold;}
```

This rule takes precedence over the one associated with the <body> element because the selector is more specific about which element it applies to.

The way in which some properties inherit saves you from having to write out rules and all the property-value pairs for each element and makes for a more compact style sheet.

6.3 Some CSS Properties

The following table shows the main properties available to you from CSS1 and CSS2:

FONT	FONT (continued)	TEXT (continued)	TEXT (continued)
font	font-variant	text-align	white-space
font-family	font-weight	text-decoration	word-spacing
font-size	TEXT	text-indent	BACKGROUND
font-size-adjust	color	text-shadow	background
font-stretch	direction	text-transform	background-attachment
font-style	letter-spacing	unicode-bidi	background-color
BACKGROUND (continued)	BORDER (continued)	DIMENSIONS (continued)	TABLE (continued)
background-image	border-top-style	min-width	table-layout
background-position	border-top-width	width	LIST and MARKER
background-repeat	border-width	POSITIONING	list-style
BORDER	MARGIN	bottom	list-style-image
border	margin	clip	list-style-position
border-bottom	margin-bottom	left	list-style-type
border-bottom-color	margin-left	overflow	marker-offset
border-bottom-style	margin-right	right	GENERATED CONTENT
border-bottom-width	margin-top	top	content
border-color	PADDING	vertical-align	counter-increment
border-left	padding	z-index	counter-reset
border-left-color	padding-bottom	OUTLINES	quotes
border-left-style	padding-left	outline	CLASSIFICATION
border-left-width	padding-right	outline-color	clear
border-right	padding-top	outline-style	cursor
border-right-color	DIMENSIONS	outline-width	display
border-right-style	height	TABLE	float
border-right-width	line-height	border-collapse	position
border-style	max-height	border-spacing	visibility
border-top	max-width	caption-side	
border-top-color	min-height	empty-cells	

6.3.1 Basic Example

The following example uses quite a number of CSS rules. The purpose of most of these rules should be clear by their name. After this example, you look at different aspects of CSS, and how to control text, tables, white space, and backgrounds.

Before starting, take a look at the XHTML document we will be working on without the CSS rules attached.

Figure 6.2 shows you what the document looks like without styling.

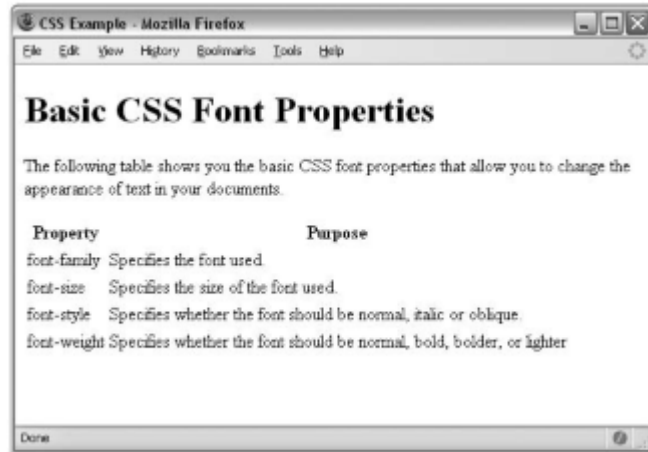


Figure 6.2: Sample page2 without CSS

Here is the code for the document you saw in Figure 6-2 (sample_page2.html). It contains a heading, a paragraph, and a table. Notice the use of the `<link>` element inside the `<head>` element, which tells the browser that this document should be styled with the style sheet specified in the value of the `href` attribute that is carried on the `<link>` element. Also note how some of the `<td>` elements carry a `class` attribute whose value is `code`; you use this to distinguish the `<td>` elements that contain code from other text in the document.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
  <head>
    <title>CSS Example</title>
    <link rel="stylesheet" type="text/css" href="sample_page2.css" />
  </head>
  <body>
    <h1>Basic CSS Font Properties</h1>
    <p>The following table shows you the basic CSS font properties
that allow you to change the appearance of text in your
documents.</p>
    <table>
      <tr>
        <th>Property</th>
        <th>Purpose</th>
```

```

</tr>
<tr>
    <td class="code">font-family</td>
    <td>Specifies the font used.</td>
</tr>
<tr>
    <td class="code">font-size</td>
    <td>Specifies the size of the font used.</td>
</tr>
<tr>
    <td class="code">font-style</td>
    <td>Specifies whether the font should be normal, italic
    or oblique.</td>
</tr>
<tr>
    <td class="code">font-weight</td>
    <td>Specifies whether the font should be normal, bold,
    bolder,
    or lighter</td>
</tr>
</table>
</body>
</html>

```

Figure 6.3 shows what this document looks like with a style sheet attached.

Now, let's take a look at the style sheet used with this document. All CSS style sheets are saved with the file extension `.css`, and this one is called `sample_page2.css`.

You should be able to create a CSS style sheet in the same editor you are using to create your XHTML pages, and because CSS files are just simple text files (like XHTML files) you can also create them in Windows Notepad or TextEdit on the Mac.

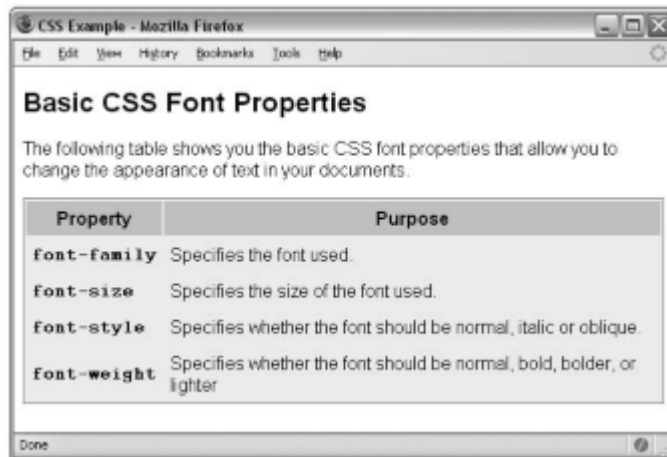


Figure 6.3: *Sample_page2* with CSS

The document is mainly separate rules, the exception being the first line — which isn't really a rule; it is a comment.

Anything between the opening `/*` and closing `*/` will be ignored by the browser and therefore will not be shown:

```
/* Style sheet for sample_page2.html */
```

The first rule applies to the `<body>` element. It specifies that the default color of any text and lines used on the page will be black, that the background of the page should be in white, and that the typeface used throughout the document should be Arial. If Arial is not available, Verdana will be used instead; failing that, any sans-serif font will be used.

```
body {
color:#000000;
background-color:#ffffff;
font-family:arial, verdana, sans-serif; }
```

There is also a `background-color` property for the body of a document because some people change the default background color of their computers (so that it is not a glaring white); if you do not set this property, the background color of those users' browsers will be whatever color they have selected.

The next two rules simply specify the size of the contents of the `<h1>` and `<p>` elements, respectively:

```
h1 {font-size:18pt;}
p {font-size:12pt;}
```

Next, it is time to add a few settings to control the appearance of the table — first to give it a light gray background, and then to draw a 1-pixel dark gray border around the edge:

```
table {
```

```
background-color:#efefef;
border-style:solid;
border-width:1px;
border-color:#999999;}
```

Within the table, the headings should have a medium gray background color (slightly darker than the main body of the table), the text should appear in a bold font, and between the edge of the cell and the text there should be 5 pixels of padding. (As you will see in more detail later in the chapter, padding is the term used for space between the edge of a box and the content inside it.)

```
th {
background-color: #cccccc;
font-weight: bold;
padding:5px;}
```

The individual table data cells have 5 pixels of padding. Adding this space makes the text much easier to read, and without it the text in one column might run up right next to the text in the neighboring column:

```
td {padding:5px;}
```

Finally, you may have noticed in Figure 6-3 that the cells of the table that mentioned CSS properties were in a Courier font. This is because the corresponding table cells in the XHTML document carried a class attribute whose value was code. On its own, the class attribute does not change the display of the document

(as you can see from Figure 6-2). The class attribute does, however, allow you to associate CSS rules with elements whose class attribute has a specific value. Therefore, the following rule applies only to <td> elements that carry a class attribute whose value is code, not to all <td> elements:

```
td.code {
font-family:courier, courier-new, serif;
font-weight:bold;}
```

Study Session Summary



Summary

In this Study Session, you learnt that

1. CSS works by allowing you to associate rules with the elements that appear in the document.
2. A CSS rule is made up of two parts: Selector and Declaration.
3. The declaration is also split into two parts, separated by a colon: a property, which is the property of the selected element(s) that

you want to affect, in this case the font-family property; a value, which is a specification for this property; in this case it is the Arial typeface.

4. CSS rules can also appear in two places inside the XHTML document:
5. Inside the <head> element, contained with a <style> element
6. As a value of a style attribute on any element that can carry the style attribute
7. One of the powerful features of CSS is that many of the properties that have been applied to one element will be inherited by child elements (elements contained within the element that the rules were declared upon).

Assessment



Assignment

- 1 What are CSS rules?
- 2 What makes up a CSS rule?
- 3 What is Inheritance?
- 4 Mention 7 CSS properties and their use.

Study Session 7

Web Programming in JavaScript (Client side)

Expected duration: 1 week or 2 contact hour

Introduction

In the previous study session, we explained CSS as a scripting language used to beautify out HTML. In this study session, we will learn how to use JavaScript on our website and the need add the client side scripting language to our website.

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 7.1 *explain* the usefulness of JavaScript
- 7.2 *state* where to include JavaScript in a document
- 7.3 *understand* the concept of variables, Constant, operators and functions
- 7.4 *explain* the terms Keywords and cite examples

Terminology

JavaScript	An object-oriented computer programming language commonly used to create interactive effects within web browsers.
Data Types	A particular kind of data item, as defined by the values it can take, the programming language used, or the operations that can be performed on it.
Variable	or scalar is a storage location paired with an associated symbolic name (an identifier), which contains some known or unknown quantity of information referred to as a value.
Operators	A character that represents an action, as for example (*) is an arithmetic operator that represents multiplication.

7.1 JavaScript

JavaScript

An object-oriented computer programming language commonly used to create interactive effects within web browsers.

Rather like CSS rules, **JavaScript** can either be embedded in a page or placed in an external script file. But in order to work in the browser, the browser must support JavaScript and must have it enabled (most browsers allow you to disable JavaScript). Bearing in mind that a user might not have JavaScript enabled in the browser, you should use JavaScript only to enhance the experience of using your pages; you should not make it a requirement in order to use or view the page.

You add scripts to your page inside the `<script>` element. The type attribute on the opening `<script>` tag indicates what scripting language will be found inside the element. There are several other scripting languages (such as VBScript or Perl), but JavaScript is by far the most popular for use in a browser.

Here you can see a very simple script that will write the words “My first JavaScript” into the mypage.html.

```
<html>
  <body>
    <p>
      <script type="text/javascript">
        document.write("My first JavaScript")
      </script>
    </p>
  </body>
</html>
```

JavaScript uses the `write()` method to write text into the document (remember that methods perform an action/calculation). The text is outputted where the script is written in the page.

Where you put your JavaScript within a page is very important. If you put it in the body of a page — as in this example — then it will run (or execute) as the page loads. Sometimes, however, you will want a script to run only when an event triggers; an event can be something like a key being pressed or a submit button being clicked. This will usually result in something known as a function being called. Functions are put inside `<script>` elements that live in the `<head>` of a page to ensure that they load before the page is displayed and are therefore ready for use immediately when the page has loaded. A function also allows you to reuse the same script in different parts of the page.

You can also write JavaScript in external documents that have the file extension `.js`. This is a particularly good option if your script is used by more than one page — because you do not need to repeat the script in

each page that uses it, and if you want to update your script you need only change it in one place. When you place your JavaScript in an external file, you need to use the `src` attribute on the `<script>` element; the value of the `src` attribute should be an absolute or relative URL pointing to the file containing the JavaScript. For example: `<script type="JavaScript" src="scripts/validation.js" />`

So there are three places where you can put your JavaScripts — and a single XHTML document can use all three because there is no limit on the number of scripts one document can contain:

- In the `<head>` of a page: These scripts will be called when an event triggers them.
- In the `<body>` section: These scripts will run as the page loads.
- In an external file: If the link is placed inside the `<head>` element, the script is treated the same as when the script lives inside the head of the document waiting for an event to trigger it, whereas if it is placed in the `<body>` element it will act like a script in the body section and execute as the page loads.

Some early browsers did not support JavaScript; therefore, you will sometimes see JavaScript written inside an HTML or XHTML comment so that older browsers can ignore the script, which would otherwise cause errors, as shown here. Newer browsers will just ignore these comments in the

`<script>` element:

```
<script type="text/javascript">
  <!--
      document.write("My first JavaScript")
  //- ->
</script>
```

Note how two forward slash characters (`//`) precede the closing characters of the XHTML comment. This is actually a JavaScript comment that prevents the JavaScript compiler from trying to process the `-- >` characters

7.2 Comments in JavaScript

Data Type

A particular kind of data item, as defined by the values it can take, the programming language used, or the operations that can be performed on it.

You can add comments to your JavaScript code in two ways. The first way, which you have already seen, allows you to comment out anything on that line after the comment marks. Here, anything on the same line after the two forward slash characters is treated as a comment:

```
<script type="text/javascript">
    document.write("My first JavaScript") // comment goes here
</script>
```

You can also comment out multiple lines using the following syntax,

holding the comment between an opening pair of characters

`/*` and a closing pair of characters `*/` like so:

```
/* This whole section is commented out so it is not treated as a part of the script. */
```

This is similar to comments in CSS.

As with all code, it's good practice to comment your code clearly, even if you are the only person likely to be using it, because what may have seemed clear when you wrote a script may not be so obvious when you come back to it later. Adding variable name descriptions and explanations of functions and their parameters are good examples of where comments make code easier to read.

7.2.1 The `<noscript>` Element

The `<noscript>` element offers alternative content for users whose browsers do not support JavaScript or have it disabled. It can contain any XHTML content that the author wants to be seen in the browser if the user does not have JavaScript enabled.

7.2.2 Data types

Different types of data (letters, whole numbers, decimal numbers, dates) are known to have different data types; these allow programs to manage the different types of data in different ways. For example, if you use the `+` operator with a string, it concatenates two strings, whereas if it is used with numbers, it adds the two numbers together. Some programming languages require that you specifically indicate what type a variable is and require you to be able to convert between types. While JavaScript supports different data types, as you are about to see, it handles conversion between types itself, so you never need to worry about telling JavaScript that a certain type of data is a date or a string (a string is a set of characters that may include letters and numbers).

7.3 Variables

Variable

or scalar is a storage location paired with an associated symbolic name (an identifier), which contains some known or unknown quantity of information referred to as a value.

Variables are used to store data. To store information in a variable, you can give the variable a name and put an equal sign between it and the value you want it to have. For example, here is a variable that contains a username:

```
userName = "Bob Stewart"
```

The variable is called `userName` and the value is Bob Stewart. If no value is given, then its value is undefined. (Note that when you are writing out the value of the variable in the code, the value is given in quotation marks.)

When you first use a variable, you are creating it. The process of creating a variable is referred to as declaring the variable. You can declare a variable with the `var` statement, like:

```
var userName = "Bob Stewart"
```

Note here that you need to use the `var` keyword only if you are creating a variable inside a function that has the same name as a global variable — although to understand this point you need to understand functions and global and local variables, which are covered later.

A variable's value can be recalled or changed by the script, and when you want to do either of these you use its name.

There are a few rules you must remember about variables in JavaScript:

- Variable names are case-sensitive.
- They must begin with a letter or the underscore character.
- Avoid giving two variables the same name within the same document as one might override the value of the other, creating an error.
- Try to use descriptive names for your variables. This makes your code easier to understand (and will help you debug your code if there is a problem with it).

7.3.1 Assigning a Value to a Variable

When you want to give a value to a variable, you put the variable name first, then an equal sign, and then on the right the value you want to assign to the variable. You have already seen values being assigned to these variables when they were declared a moment ago. So, here is an example of a variable being assigned a value and then the value being changed:

```
var userName = "Bob Stewart"
```

```
userName = "Robert Stewart"
```

`userName` is now the equivalent of Robert Stewart.

7.3.2 Lifetime of a Variable

When you declare a variable in a function it can be accessed only in that function. (As promised, you will learn about functions shortly.) After the function has run, you cannot call the variable again. Variables in functions are called local variables.

Because a local variable works only within a function, you can have different functions that contain variables of the same name (each is recognized by that function only).

If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared and ends when the page is closed.

Local variables take up less memory and resources than page-level variables because they require only the memory during the time that the function runs, rather than having to be created and remembered for the life of the whole page.

7.4 Operators

Operator

A character that represents an action, as for example (*) is an arithmetic operator that represents multiplication.

The **operator** itself is a keyword or symbol that does something to a value when used in an expression. For example, the arithmetic operator + adds two values together.

The symbol is used in an expression with either one or two values and performs a calculation on the values to generate a result. For example, here is an expression that uses the x operator:

area = (width x height)

An expression is just like a mathematical expression. The values are known as operands. Operators that require only one operand (or value) are sometimes referred to as unary operators, while those that require two values are sometimes called binary operators.

The different types of operators, namely:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- String operators

7.4.1 Arithmetic Operators

Arithmetic operators perform arithmetic operations upon operands. (Note that in the examples in the following table, x = 10.)

Symbol	Description	Example (x = 10)	Result
+	Addition	x+5	15
-	Subtraction	x-2	8
*	Multiplication	x*3	30
/	Division	x/2	15
%	Modulus (division remainder)	x%3	1
++	Increment (increments the variable by 1 — this technique is often used in counters)	x++	11
--	Decrement (decreases the variable by 1)	x--	9

Figure 7.1: Arithmetic Operators

7.4.2 Assignment Operators

The basic assignment operator is the equal sign, but do not take this to mean that it checks whether two values are equal. Rather, it's used to assign a value to the variable on the left of the equal sign, as you have seen in the previous section that introduced variables.

The assignment operator can be combined with several other operators to allow you to assign a value to a variable and perform an operation in one step. For example, with the arithmetic operators, the assignment operators

can be used to create shorthand versions of operators, as in the following statement:

```
total = total - profit
```

This can be reduced to the following statement:

```
total -= profit
```

While it might not look like much, this kind of shorthand can save a lot of code if you have a lot of calculations such as this (see table below) to perform.

Symbol	Example Using Shorthand	Equivalent Without Shorthand
+=	x+=y	x=x+y
-=	x-=y	x=x-y
=	x=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Figure 7.2: Comparison Operators

7.4.3 Comparison Operators

As you can see in the table that follows, comparison operators compare two operands and then return either true or false based on whether the comparison is true or not.

Note that the comparison for checking whether two operands are equal is two equal signs (a single equal sign would be an assignment operator).

Operator	Description	Example
==	Equal to	1==2 returns false 3==3 returns true
!=	Not equal to	1!=2 returns true 3!=3 returns false
>	Greater than	1>2 returns false 3>3 returns false 3>2 returns true
<	Less than	1<2 returns true 3<3 returns false 3<1 returns false
>=	Greater than or equal to	1>=2 returns false 3>=2 returns true 3>=3 returns true
<=	Less than or equal to	1<=2 returns true 3<=3 returns true 3<=4 returns false

Figure 7.3: Arithmetic Operators

7.4.3 Logical or Boolean Operators

Logical or Boolean operators return one of two values: true or false. They are particularly helpful because they allow you to evaluate more than one expression at a time. The two operands in a logical or Boolean operator

evaluate to either true or false. For example, if $x=1$ and $y=2$, then $x<2$ is true and $y>1$ is true. So the following expression: $(x<2 \ \&\& \ y>1)$ returns true because both of the operands evaluate to true.

7.4.4 String Operator

You can also add text to strings using the $+$ operator. For example, here the $+$ operator is being used to add two variables that are strings together:

```
firstName = "Bob"
```

```
lastName = "Stewart"
```

```
name = firstName + lastName
```

The value of the name variable would now be Bob Stewart. The process of adding two strings together is known as concatenation.

You can also compare strings using the comparison operators you just met. For example, you could check whether a user has entered a specific value into a text box.

There are three simple data types in JavaScript:

- Number: Used to perform arithmetic operations (addition, subtraction, multiplication, and division).
- Any whole number or decimal number that does not appear between quotation marks is considered a number.
- String: Used to handle text. It is a set of characters enclosed by quotation marks.
- Boolean: A Boolean value has only two possible values: true and false. This data allows you to perform logical operations and check whether something is true or false. You may well come across two other data types:
- Null: Indicates that a value does not exist. This is written using the keyword null. This is an important value because it explicitly states that no value has been given. This can mean a very different thing from a string that just contains a space or a zero.
- Undefined: Indicates a situation where the value has not been defined previously in code and uses the JavaScript keyword undefined. You might remember that if you declare a variable but do not give it a value, the variable is said to be undefined (you are particularly likely to see this when something is not right in your code).

7.5 Keywords

There are several keywords in JavaScript that perform functions, such as break, for, if, and while, all of which have special meaning; therefore, these words should not be used in variable, function, method, or object names. The following is a list of the keywords that you should avoid using (some of these are not actually used yet, but are reserved for future use):

abstract, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, extends, false, final, finally, float, for, function,

goto, if, implements, import, in, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, var, void, while, with.

If you are working on a page that contains more than one scripting language, in order to indicate the default scripting language, a `<meta>` element should be used in the `<head>` of the document.

```
<meta http-equiv="Content-Script-Type" content="text/JavaScript">
```

7.6 Function and Function Call

A function is some code that is executed when an event fires or a call to that function is made, and typically a function contains several lines of code. Functions are either written in the `<head>` element and can be reused in several places within the page, or in an external file that is linked from inside the `<head>` element.

There are three parts to creating or defining a function:

- Define a name for it.
- Indicate any values that might be required as arguments.
- Add statements.

For example, if you want to create a function to calculate the area of a rectangle, you might name the function `calculateArea()` (remembering a function name should be followed by parentheses). Then in order to calculate the area, you need to know the rectangle's width and height, so these would be passed in as arguments (arguments are the information the function needs to do its job). Inside the function between the curly braces are the statements, which indicate that area is equal to the width multiplied by the height (both of which have been passed into the function). The area is then returned.

```
function calculateArea(width, height) {  
    area = width * height  
    return area  
}
```

If a function has no arguments it should still have parentheses after its name; for example, `logout()`.

The `calculateArea()` function does nothing sitting on its own in the head of a document; it has to be called. In this example, you can call the function from a simple form using the `onclick` event, so that when the user clicks the Submit button the area will be calculated.

Here you can see that the form contains two text inputs for the width and height, and these are passed as arguments to the function like:

```
<form name="frmArea" action="">
```

```
Enter the width and height of your rectangle to calculate the size:<br />
Width: <input type="text" name="txtWidth" size="5" /><br />
Height: <input type="text" name="txtHeight" size="5" /><br />
<input type="button" value="Calculate area"
onclick="alert(calculateArea(document.frmArea.txtWidth.value,
document.frmArea.txtHeight.value))" />
</form>
```

7.6.1 The Return Statement

Functions that return a result must use the return statement. This statement specifies the value that will be returned to where the function was called. The calculateArea() function, for example, returned the area of the rectangle:

```
function calculateArea(width, height) {
    area = width * height
    return area
}
```

Some functions simply return true or false values. When you look at events later in the chapter, you will see how a function that returns false can stop an action from occurring. For example, if the function associated with an onsubmit event on a form returns false, the form is not submitted to the server.

7.7 Practical Tips for Writing Scripts

Before you start looking at the examples, there are a few practical hints on developing JavaScripts that should save time.

7.7.1 Online Script

Of course, some tasks will require that you create your own scripts, but if there is a script already written that you can use, then there's no point reinventing the wheel; you should consider just using that script.

Here are a couple of sites that will help you get going (and don't forget you can search using a search engine such as Google, too):

- www.HotScripts.com
- www.JavaScriptKit.com
- <http://JavaScript.Internet.com>

Even if you do not copy the script exactly, you can learn a lot by looking at how someone else has approached the same task.

7.7.2 Reusable Functions

Along with reusing other people’s scripts and folders, you should also write code that you can reuse yourself. So, you should aim to make your functions as reusable as possible rather than tying each script into the one page.

7.7.3 Using External JavaScript Files

Whenever you are going to use a script in more than one page it’s a good idea to place it in an external JavaScript file (a technique you learned about at the beginning of Chapter 11). For example, in the “Image Rollovers” section later in the chapter you will see an example of a script that creates image rollovers for a navigation bar. Your navigation will appear on each page, so rather than including the image rollover function in each page, you can just include the one script into every page. This has the following three advantages:

- If you need to change something about the navigation, you need to change only the one function, not every page.
- The file size of the pages is smaller because the JavaScript is in one file that is included on each page rather than repeated.
- You do not have to copy and paste the same code into several files.

7.7.4 Place Scripts in a Scripts Folder

When you use external scripts you should create a special scripts folder—just as you would an images folder. This helps improve the organization of your site and your directory structure. Whenever you need to look at or change a script, you know exactly where it will be.

You should also use intuitive names for your script files so that you can find them quickly and easily.

7.8 Form Validation

Validation can happen in two places, either in the browser using JavaScript or on the server. The reason for the validation on the browser is that it helps the user enter the correct data required for the job without the form being sent to the server, being processed, and then being sent back again if there are any errors. It’s much quicker to force the user to fix errors before submitting the form to the server. The server then double checks before passing the form data onto another part of the application— this second level of validation is performed because a simple wrong value in a database could prevent the application from running properly, and if the user does not have JavaScript enabled, then the application will not be compromised by the user’s submitting a value that has not been checked using JavaScript in the browser.

Forms are usually validated using the onsubmit event handler, which triggers a validation function stored in the head of the document (or in an

external file that is specified in the head of the document), so the values are checked when the user presses the Submit button. The function must then return true in order for the form to be sent. If an error is encountered, the function returns false and the user's form will not be sent — at which point the form should indicate to the user where there is a problem with what the user entered on the form.

The onsubmit event will often call a function with a name along the lines of validate(form) or validateForm(form). Because many forms contain several controls that require validation, you do not usually pass the values of each item you are checking into a validation function. The function is usually written explicitly for that form — although you can reuse the techniques you have learned in different forms (or even reuse entire functions for login or registration forms).

The first task in a validation function is to set a variable for the return value of the function to be true. Then the values entered are checked, and whenever the function finds an error in what the user has entered, this value can be turned to false to prevent the form from being submitted.

7.8.1 Checking Text Fields

You have probably seen forms on web sites that ask you to provide a username and password, and then to re-enter the password to make sure you did not mistype something. It might resemble Figure 7-1.



Figure 7.4: Validating Text fields

In such a form you might want to check a few things:

- That the username is of a minimum length
- That the password is of a minimum length
- That the two passwords match

The validate() function you are about to look at will live between the following <script> tags in the head of the document (remember, if you were going to reuse the function on other pages it could live in an external JavaScript file):

```
<script type="text/JavaScript">  
</script>
```

To start, the validation() function assigns a variable called returnValue to true; if no errors are found this will be the value that the function returns,

which will allow the form to be sent. Then the form collects the values of the form controls into variables, as follows:

```
function validate(form) {  
    var returnValue = true;  
    var username = frmRegister.txtUserName.value;  
    var password1 = frmRegister.txtPassword.value;  
    var password2 = frmRegister.txtPassword2.value;
```

The first thing you want to do is check whether the username is at least six characters long:

```
if(username.length < 6) {  
    returnValue = false;  
    alert("Your username must be at least\n6 characters long.\nPlease try again.");  
    frmRegister.txtUserName.focus();  
}
```

The length property of the username variable is used to check whether the length of the username entered is longer than six characters. If it is not, the return value of the function will be false, the form will not be submitted, and the user will see an alert box with the specified error message. Note how the focus is passed back to the form control that has a problem using the focus() method on this control, saving the user from looking through the form to find that entry again. You can also see from this example how the line break is used in the alert box to indicate breaks in the message presented to the user \n.

Next you want to check the length of the first password —this uses the same approach but also sets both of the password boxes to blank again if the password is not long enough and gives focus to the first password box:

```
if (password1.length < 6) {  
    returnValue = false;  
    alert("Your password must be at least\n6 characters long.\nPlease try again.");  
    frmRegister.txtPassword.value = "";  
    frmRegister.txtPassword2.value = "";  
    frmRegister.txtPassword.focus();  
}
```

If the code has gotten this far, the username and first password are both long enough. Now, you just have to check whether the value of the first password box is the same as the second one, as shown here. Remember that the != operator used in this condition means “not equal”:

```

if (password1.value != password2.value) {
returnValue = false;
alter("Your password entries did not match.\nPlease try again.");
frmRegister.txtPassword.value = "";
frmRegister.txtPassword2.value = "";
frm Register.txtPassword.focus();
}

```

You can see here that when the user has entered passwords that do not match, the user is shown an alert box with an error message reporting that the password entries did not match. Also the contents of both password inputs are cleared and the focus is passed back to the first password box.

When the user makes a mistake with a password input, there is no point in leaving values in the password form controls because users will not be able to see the values they have entered into these boxes (because it will show dots or asterisks rather than the characters). Therefore, users will have to enter both values again because they will not be able to see where the error is.

The only thing left to do is return the value of the returnValue variable — which will be true if all the conditions are met or false if not.

```

return returnValue;
}

```

Here is the form that is used with this example:

```

<form name="frmRegister" method="post" action="register.aspx"
onsubmit="return validate(this);">
    <div class="label"><label
for="txtUsername">Username:</label></div>
    <div class="formElement">
    <input type="text" name="txtUserName" id="txtUserName"
size="12" />
    </div>
    <div class="label"><label for="txtPassword">Password:
</td></label></div>
    <div class="formElement">
    <input type="password" name="txtPassword" id="txtPassword"
size="12" />
    </div>
    <div class="label"><label for="txtPassword2">Confirm your
password:</label></div>
    <div class="formElement">

```

```

        <input type="password" name="txtPassword2"
id="txtPassword2" size="12" />
    </div>
    <div class="label">&nbsp;</div>
    <div class="formElement"><input type="submit" value="Log
in" /></div>
</form>

```

In Figure 7.5 you can see the result if the user's password is not long enough.



Figure 7.5: Validating Text fields

7.8.2 Required Text Fields

Often you will want to ensure that a user has entered some value into a text field. You can do this for an individual element using the technique you saw in the last example for the username. As you saw then, if users entered a value that was less than six characters long they were alerted, and the form would not submit.

You can see this function working with a form that is very similar to the one in the last example, although the values for the name attributes have to be descriptive for the user and match the labels for those forms

```

<form name="frmEnquiry" method="post" action="register.aspx"
    onsubmit="return validate(this);">
    <div class="label"><label for="Name">Name:</div>
    <div class="formElement">
        <input type="text" class="required" name="Name" size="12"
id="Name" />

```



```

</div>
<div class="label"><label for="E-mail">E-mail:</div>
<div class="formElement">
  <input type="text" class="required" name="E-mail" size="12"
    id="E-mail" />
</div>
<div class="label"><label for="txtEmail">Please enter your query
here:</div>
<div class="formElement">
  <textarea rows="8" class="required" cols="30" name="Query"
    id="Query">
  </textarea>
</div>
<div class="label"><label for="txtEmail">&nbsp;</div>
<div class="formElement">
  <input type="submit" class="" value="Submit your query" />
</div>
</form>

```

Figure 7.6 shows the error message generated when the user has not entered a value for the e-mail address. The word e-mail in quotes has been retrieved from the name attribute of that text input.

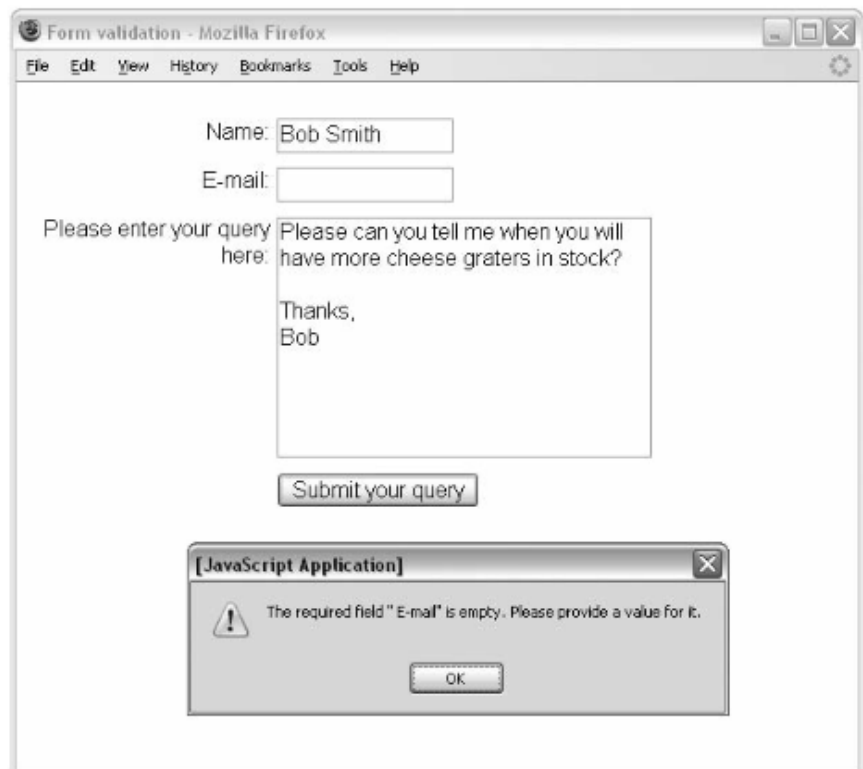


Figure 7.6: Required Text field

7.8.3 Preventing a Form Submission Until a Checkbox Has Been Selected

If you want to ensure that a checkbox has been selected — for example, if you want a user to agree to certain terms and conditions — you can do so by adding a function to the onsubmit event handler similar to those you have seen already.

The function checks whether the checkbox has been checked, and if

The function returns true the form will be submitted. If the function returns false, the user would be prompted to check the box:

```
function checkCheckBox(myForm){
    if (myForm.agree.checked == false )
    {
        alert('You must agree to terms and conditions to
continue');
        return false;
    } else
        return true;
}
```

Another common technique is to use script to simply disable the Submit button until users have clicked the box to say that they agree with the terms and conditions.

The following is a very simple page with a form. When the page loads, the Submit button is disabled in the onload event. If the user clicks the chkAgree checkbox, then the Submit button will be re-enable:

```
<body onload="document.frmAgree.btnSubmit.disabled=true">
<form name="frmAgree" action="test.aspx" method="post">
I understand that this software has no liability:
<input type="checkbox" value="0" name="chkAgree" id="chkAgree"
onclick="document.frmAgree.btnSubmit.disabled=false" />
<input type="submit" name="btnSubmit" value="Go to download" /><br
/>
<p>You will not be able to submit this form unless you agree to the
<a href="terms.html">terms and conditions</a> and check the terms and
conditions box.</p>
```

</form>

</body>

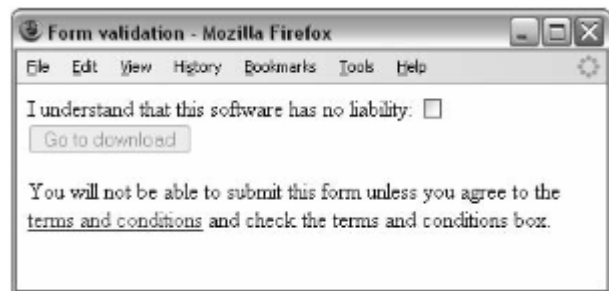


Figure 7.7: Disabling form submission

7.8.4 Testing Characters Using Test and Regular Expressions

Regular Expressions can also be used to test patterns of strings entered by users. For example, they can be used to test whether there are any spaces in a string, whether the string follows the format of an e-mail address, whether it's an amount of currency, and so on. This uses the test() method like so: first you set variables to hold the return value of true, the value entered by a user, and a value to hold the Regular Expression.

```
function validate(form) {  
    var returnValue = true;  
    var amountEntered = document.frmCurrency.txtAmount.value;  
    var currencyFormat = /^ \d+(\.\d{1,2})?$/;
```

Then you test whether the value follows the correct format — if it does not, you alert the user, give focus back to the correct form element, and set the returnValue variable to false:

```
    if (currencyFormat != test(amountEntered))  
    {  
        alert("You did not enter an amount of money");  
        document.frmCurrency.txtAmount.focus();  
        returnValue = false;  
    }  
    return returnValue;  
}
```

Here is the simple form to test this example:

```
<form name="myForm" onsubmit="return validate(this);"  
action="money.aspx" method="get">
```

Enter an amount of money here \$

```
<input type="text" name="txtAmount" id="txtAmount" size="7" />
```

```
<input type="submit" value="Check format" />
</form>
```

The following table that follows lists some helpful regular expressions that you can use to get you started:

Test for	Description	Regular Expression
White space	No white-space characters.	<code>\S/;</code>
Alphabetic characters	No characters of the alphabet nor the hyphen, period, or comma may appear in the string.	<code>/[^a-z \ -\ .']/gi;</code>
Alphanumeric characters	No letters or number may appear in the string.	<code>/[^a-z0-9]/gi;</code>
Credit card details	A 16-digit credit card number following the pattern XXXX XXXX XXXX XXXX.	<code>/^\d{4}([-]?\d{4}){3}\$/;</code>
Decimal number	A number with a decimal place.	<code>/^\d+(\.\d+)?\$/;</code>
Currency	A group of one or more digits followed by an optional group consisting of a decimal point plus one or two digits.	<code>/^\d+(\.\d{1,2})?\$/;</code>
E-mail address	An e-mail address.	<code>/^\w(\.?\w-)*@\w(\.?\w-)*\.[a-z]{2,6}(\.[a-z]{2})?\$/i;</code>

Figure 7.8: Examples of regular expressions

7.9 When Not To Use JavaScript

7.9.1 Drop-Down Navigation Menus

One of the more common requests is for drop-down navigation menus where subpages drop down from the main items on the menu. These rely on JavaScript, and I discourage clients from using them for three reasons:

- The technique simply will not work for those who have JavaScript turned off on their browser. While this is quite a small percentage, it does mean that those users simply cannot access those pages.
- The technique tends to perform slightly differently on different browsers, and it's hard to get a script to work on all browsers.
- Users can find it difficult to click the appropriate part of a menu that moves (especially if they have a disability or a sticky mouse).

7.9.2 Hiding Your E-mail Address

Several articles on the Web that suggest you can use JavaScript to write your e-mail address to pages (using the `write()` method of the `document` object to write out the e-mail address, rather than a normal `<a>` link and XHTML). The goal is to avoid getting so much spam. Among sources for spam are little programs (that often go under the name of bots, spiders, or

crawlers) that crawl through web sites looking for e-mail addresses. These e-mail addresses are then used as a target for spam. The problem with this idea is that anyone without JavaScript turned on in their browser will not be able to see your e-mail address.

A better alternative is to provide an e-mail form that sends inquiries to you — then once you have received an inquiry you can be fairly sure the user will not be doing this just to get an e-mail address and that you are safe giving your e-mail address to this user.

7.9.3 Quick Jump Select Boxes

Some sites offer select boxes in forms as a navigation menu —often referred to as a quick jump menu that takes you directly to different pages or sections of the site when you select that item from the drop-down list box. Some of these use scripts to automatically take the user to the selected page without the user’s actually pressing a GO or Submit button. Rather, the script is set up to detect a change in the select box and then to take the user to that page. This is bad practice for two reasons:

- You can use the up and down arrow keys to select items from a select box, and any user who tried to do this would automatically get taken to the first selection as soon as he or she pressed the down arrow the first time. Users would never be able to get further than this option using keys. While a savvy user might pick this up quickly, those with disabilities who are using keys rather than a mouse to navigate the site might be a lot more frustrated.
- And again, if the user has JavaScript disabled, it simply won’t work.

7.9.4 Anything the User Requires from Your Site

The bottom line in the decision on using JavaScript is whether it will simply enhance the user experience or whether it is required for the user to perform an action or see some vital information. You should never design anything that requires JavaScript in order to function —remember the lesson from the “Disabling a Submit Button until a Checkbox Has Been Selected” section.

Study Session Summary



Summary

In this Study Session, we discussed that

1. JavaScript is by far the most popular script for use in a browser amongst other scripting languages such as VBScript or Perl.
2. Where you put your JavaScript within a page is very important. If you put it in the body of a page — as in this example — then it will run (or execute) as the page loads.
3. Variables are named memory location used to store data. The type of value a variable store is called Data type.

-
4. Keywords are reserved words that perform functions, such as break, for, if, and while, all of which have special meaning.
 5. The operator itself is a keyword or symbol that does something to a value when used in an expression. There are four types: arithmetic, logical, String, comparison.
 6. An expression is just like a mathematical expression. The values are known as operands.
 7. Validation can happen in two places, either in the browser using JavaScript or on the server.
 8. The reason for the validation on the browser is that it helps the user enter the correct data.
 9. Forms are usually validated using the onsubmit event handler
 10. One of the key things to remember, however, is that you should use JavaScript to enhance a page, rather than relying on it to display content or offer some functionality.

Assessment



Assignment

1. What are variables, data types and constants?
2. Mention the various Life scopes of variables that you know.
3. Mention the types of operators you know, with examples
4. What are functions?
5. Why do we need to use functions?
6. Write a JavaScript to compute the age of a student who enters his/her date of birth.

Study Session 8

Web Programming in PHP (Server side)

Expected duration: 1 week or 2 contact hour

Introduction

In the previous study session, we explained JavaScript as an object-oriented computer programming language commonly used to create interactive effects within web browsers. In this study session, we will learn how to use the server side scripting language to add dynamic effect to our website. Such effects includes server side form validation, transmitting information or data from page to page and also accessing the database.

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 8.1 *examine* the difference between tags, elements and attributes
- 8.2 outline how a web page uses markup to describe how the page should be structured
- 8.3 use the elements that allow you to mark up text such as headings and paragraphs
- 8.4 use elements that can add additional presentation information and phrasing to your documents
- 8.5 add bulleted and numbered lists to documents
- 8.6 distinguish different types of elements in XHTML

Terminology

PHP	Acronym for hypertext pre-processor, its defined as an HTML-embedded scripting language that is used to write web pages.
PHP Constant	A constant is an identifier (name) for a simple value. The value cannot be changed during the script. A valid constant name starts with a letter or underscore (no \$ sign before the constant name). Note: Unlike variables, constants are automatically global across the entire script.
Operator	This a property that determines how operators of the same

Precedence

precedence are grouped in the absence of parentheses.

8.1 PHP: Hypertext Preprocessor

PHP

or hypertext preprocessor is defined as an HTML-embedded scripting language that is used to write web pages.

PHP is a recursive acronym for "PHP: Hypertext Preprocessor".

Common uses of PHP:

PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them. The other uses of PHP are:

PHP can handle forms, i.e. gather data from files, save data to a file, thru email

- You can send data, return data to the user.
- You add, delete, modify elements within your database thru PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

8.2 Escape to PHP

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP.' There are four ways to do this:

Canonical PHP tags

The most universally effective PHP tag style is:

```
<?php...?>
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

Short-open (SGML-style) tags

Short or short-open tags look like this:

```
<?...?>
```

Short tags are, as one might expect, the shortest option You must do one of two things to enable PHP to recognize the tags:

- Choose the `--enable-short-tags` configuration option when you're building PHP.
- Set the `short_open_tag` setting in your `php.ini` file to `on`. This option must be
- disabled to parse XML with PHP because the same syntax is used for XML tags.

ASP-style tags

ASP-style tags mimic the tags used by Active Server Pages to delineate code blocks. ASPstyle tags look like this:

```
<%...%>
```


To use ASP-style tags, you will need to set the configuration option in your php.ini file.

HTML script tags

HTML script tags look like this:

```
<script language="PHP">...</script>
```

"Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this:

```
<html>
  <head>
    <title>Hello World</title>
  <body>
    <?php echo "Hello, World!";?>
  </body>
</html>
```

It will produce following result:

Hello, World!

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

8.3 Commenting PHP Code

A comment is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

Single-line comments: They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

```
<?
# This is a comment, and
# This is the second line of the comment
// This is a comment too. Each style comments only
print "An example with single line comments";
```

```
?>
```

Multi-lines printing: Here are the examples to print multiple lines in a single print statement:

```
<?
```

```
# First Example
```

```
print <<<END
```

This uses the "here document" syntax to output multiple lines with \$variable interpolation. Note that the here document terminator must appear on a line with just a semicolon no extra whitespace!

```
END;
```

```
# Second Example
```

```
print "This spans
```

```
multiple lines. The newlines will be  
output as well";
```

```
?>
```

Multi-lines comments: They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here is the example of multi lines comments.

```
<?
```

```
/* This is a comment with multiline
```

```
Author : Mohammad Mohtashim
```

```
Purpose: Multiline Comments Demo
```

```
Subject: PHP
```

```
*/
```

```
print "An example with multi line comments";
```

```
?>
```

8.4 PHP is whitespace insensitive

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row. One whitespace character is the same as many such characters.

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable \$four is equivalent:

```
$four = 2 + 2; // single spaces
```

```
$four <tab>=<tab2<tab>+<tab>2 ; // spaces and tabs
```

```
$four =
```

```
2+
```

```
2; // multiple lines
```

8.5 PHP is case sensitive

Yeah it is true that PHP is a case sensitive language. Try out following example:

```
<html>
  <body>
    <?
      $capital = 67;
      print("Variable capital is $capital<br>");
      print("Variable CaPiTaL is $CaPiTaL<br>");
    ?>
  </body>
</html>
```

This will produce following result:

```
Variable capital is 67
```

```
Variable CaPiTaL is
```

8.6 Statements are Expressions Terminated by Semicolons

A statement in PHP is any expression that is followed by a semicolon (;). Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called \$greeting:

```
$greeting = "Welcome to PHP!";
```

Expressions are combinations of tokens

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables (\$two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth.

Braces make blocks

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent:

```
if (3 == 2 + 1)
    print("Good - I haven't totally lost my mind.<br>");
if (3 == 2 + 1)
{
    print("Good - I haven't totally");
    print("lost my mind.<br>");
}
```

8.7 Variables

The main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign (\$).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

8.8 Data types

PHP has a total of eight data types which we use to construct our variables:

- Integers: are whole numbers, without a decimal point, like 4195.
\$int_var = 12345;
\$another_int = -12345 + 12345;
- Doubles: are floating-point numbers, like 3.14159 or 49.1.
\$many = 2.2888800;
\$many_2 = 2.2111200;
\$few = \$many + \$many_2;
print(.\$many + \$many_2 = \$few
.);
- Booleans: have only two possible values either true or false.

if (TRUE)

```
print("This will always print<br>");  
else  
print("This will never print<br>");
```

- NULL: is a special type that only has one value: NULL.
\$my_var = NULL;
- Strings: are sequences of characters, like 'PHP supports string operations.'
\$string_1 = "This is a string in double quotes";
\$string_2 = "This is a somewhat longer, singly quoted string";
\$string_39 = "This string has thirty-nine characters";
\$string_0 = ""; // a string with zero characters

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with \$) are replaced with string representations of their values.

The escape-sequence replacements are:

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \\$ is replaced by the dollar sign itself (\$)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

- Arrays: are named and indexed collections of other values.
- Objects: are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- Resources: are special variables that hold references to resources external to PHP (such as database connections).

The first five are simple types, and the next two (arrays and objects) are compound – the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

8.9 Constant

PHP Constant

A constant is an identifier (name) for a simple value.

The value cannot be changed during the script.

A valid constant name starts with a letter or underscore (no \$ sign before the constant name).

Note: Unlike variables, constants are automatically

A **constant** is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use define() function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a \$. You can also use

global across the entire script.

the function `constant()` to read a constant's value if you wish to obtain the constant's name dynamically.

`constant()` function

As indicated by the name, this function will return the value of the constant.

This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function.

`constant()` example

```
<?php
define("MINSIZE", 50);
echo MINSIZE;

echo constant("MINSIZE"); // same thing as the previous line
?>
```

Only scalar data (boolean, integer, float and string) can be contained in constants.

Differences between constants and variables are

- There is no need to write a dollar sign (\$) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the `define()` function.
- Constants may be defined and accessed anywhere without regard to variable scoping rules.
- Once the Constants have been set, may not be redefined or undefined.

8.10 Operator Types

PHP language supports following type of operators.

- Arithmetic Operators: +, -, *, /, --, ++, %

```
<html>
<head><title>Arithmetical Operators</title></head>
<body>
<?php
$a = 42;
$b = 20;
$c = $a + $b;
echo "Addition Operation Result: $c <br/>";
$c = $a - $b;
echo "Subtraction Operation Result: $c <br/>";
$c = $a * $b;
echo "Multiplication Operation Result: $c <br/>";
$c = $a / $b;
```

```

echo "Division Operation Result: $c <br/>";
$c = $a % $b;
echo "Modulus Operation Result: $c <br/>";
$c = $a++;
echo "Increment Operation Result: $c <br/>";
$c = $a--;
echo "Decrement Operation Result: $c <br/>";
?>
</body>
</html>

```

This will produce the following result:

Addition Operation Result: 62

Subtraction Operation Result: 22

Multiplication Operation Result: 840

Division Operation Result: 2.1

Modulus Operation Result: 2

Increment Operation Result: 42

Decrement Operation Result: 43

- Comparison Operators: >=, <=, <, >, ==, !=

```

<html>
<head><title>Comparison Operators</title><head>
<body>
<?php
$a = 42;
$b = 20;
if( $a == $b ){
echo "TEST1 : a is equal to b<br/>";
}else{
echo "TEST1 : a is not equal to b<br/>";
}
if( $a > $b ){
echo "TEST2 : a is greater than b<br/>";
}else{
echo "TEST2 : a is not greater than b<br/>";
}
if( $a < $b ){
echo "TEST3 : a is less than b<br/>";
}else{
echo "TEST3 : a is not less than b<br/>";
}
if( $a != $b ){
echo "TEST4 : a is not equal to b<br/>";
}else{
echo "TEST4 : a is equal to b<br/>";
}
if( $a >= $b ){
echo "TEST5 : a is either greater than or equal to b<br/>";
}else{
echo "TEST5 : a is neither greater than nor equal to b<br/>";
}
if( $a <= $b ){
echo "TEST6 : a is either less than or equal to b<br/>";
}

```

```

    }else{
    echo "TEST6 : a is neither less than nor equal to b<br/>";
    }
    ?>
</body>
</html>

```

This will produce the following result:

```

TEST1 : a is not equal to b
TEST2 : a is greater than b
TEST3 : a is not less than b
TEST4 : a is not equal to b
TEST5 : a is either greater than or equal to b
TEST6 : a is neither less than nor equal to b

```

- Logical (or Relational) Operators: and, or, &&, ||, !

```

<html>
<head><title>Logical Operators</title><head>
<body>
<?php
$a = 42;
$b = 0;
if( $a && $b ){
echo "TEST1 : Both a and b are true<br/>";
}else{
echo "TEST1 : Either a or b is false<br/>";
}
if( $a and $b ){
echo "TEST2 : Both a and b are true<br/>";
}else{
echo "TEST2 : Either a or b is false<br/>";
}
if( $a || $b ){
echo "TEST3 : Either a or b is true<br/>";
}else{
echo "TEST3 : Both a and b are false<br/>";
}
if( $a or $b ){
echo "TEST4 : Either a or b is true<br/>";
}else{
echo "TEST4 : Both a and b are false<br/>";
}
$a = 10;
$b = 20;
if( $a ){
echo "TEST5 : a is true <br/>";
}else{
echo "TEST5 : a is false<br/>";
}
if( $b ){
echo "TEST6 : b is true <br/>";
}else{
echo "TEST6 : b is false<br/>";
}

```



```

}
if( !$a ){
echo "TEST7 : a is true <br/>";
}else{
echo "TEST7 : a is false<br/>";
}
if( !$b ){
echo "TEST8 : b is true <br/>";
}else{
echo "TEST8 : b is false<br/>";
}
?>
</body>
</html>

```

- Assignment Operators: =, *=, /=, +=, -=, %=

```

<html>
<head><title>Assignment Operators</title></head>
<body>
<?php
$a = 42;
$b = 20;
$c = $a + $b; /* Assignment operator */
echo "Addition Operation Result: $c <br/>";
$c += $a; /* c value was 42 + 20 = 62 */
echo "Add AND Assignment Operation Result: $c <br/>";
$c -= $a; /* c value was 42 + 20 + 42 = 104 */
echo "Subtract AND Assignment Operation Result: $c <br/>";
$c *= $a; /* c value was 104 - 42 = 62 */
echo "Multiply AND Assignment Operation Result: $c <br/>";
$c /= $a; /* c value was 62 * 42 = 2604 */
echo "Division AND Assignment Operation Result: $c <br/>";
$c %= $a; /* c value was 2604/42 = 62*/
echo "Modulus AND Assignment Operation Result: $c <br/>";
?>
</body>
</html>

```

This will produce the following result:

```

Addition Operation Result: 62
Add AND Assignment Operation Result: 104
Subtract AND Assignment Operation Result: 62
Multiply AND Assignment Operation Result: 2604
Division AND Assignment Operation Result: 62
Modulus AND Assignment Operation Result: 20

```

- Conditional (or ternary) Operators: ?:

It first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

```

<html>
<head><title>Arithmetical Operators</title></head>
<body>
<?php
$a = 10;

```

```

$b = 20;
/* If condition is true then assign a to result otherwise b */
$result = ($a > $b) ? $a : $b;
echo "TEST1 : Value of result is $result<br/>";
/* If condition is true then assign a to result otherwise b */
$result = ($a < $b) ? $a : $b;
echo "TEST2 : Value of result is $result<br/>";
?>
</body>
</html>

```

This will produce the following result:

TEST1 : Value of result is 20

TEST2 : Value of result is 10

8.11 Operators Categories

Operator Precedence

A characteristic of operators that indicates when they will be evaluated when they appear in complex expressions.

Operators with high precedence are evaluated before operators with low precedence. For example, the multiplication operator (*) has higher preference than the addition operator (+), so the expression

$2+3*4$ equals 14, not 20.

All the operators we have discussed above can be categorized into following categories:

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

8.11.1 Precedence of PHP Operators

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

Figure 8.1: PHP operator precedence

An array is a data structure that stores one or more similar type of values in a single value.

For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID which is called array index.

- Numeric array - An array with a numeric index. Values are stored and accessed in linear fashion
- Associative array - An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- Multidimensional array - An array containing one or more arrays and values are accessed using multiple indices

NOTE: Built-in array functions is given in function reference PHP Array Functions

8.12 PHP Code

PHP code starts out looking like HTML code and ends up looking nothing like it. If you've looked at HTML code, you've seen things that look like "" or "<h2>". You will put your PHP code between "<?" and "?>".

For example, the following web page will display the current time:

```
<html>
  <head>
    <title>My PHP Page</title>
  </head>
  <body>
    <h1>PHP Test Page</h1>
    <p>The current time is <?echo date('h:i A')?></p>
  </body>
</html>
```

Except for the php part, this looks like a normal web page. The only PHP part is the “<?echo date('h:i A')?>”. If you save this file as “test.php” on your web site, and then view it, you will see the current time every time you reload your page.

8.13 Functions

PHP uses functions to get things done. The functions we just used are the “echo” function and the “date” function. The ‘echo’ function is simple. Whatever you give it, it ‘echoes’ to the web page.

In this case, we gave it the ‘date’ function. The ‘date’ function returns the current date and time in whatever format you want. The format code we used was “h:i A”. This means we want the hour, a colon, the minute, a space, and either AM or PM. If it is currently March 9, 10:11 AM and 14 seconds, this will give us “10:11 AM”, which the “echo” function inserts as part of the web page.

8.14 PHP with Forms

PHP is designed to be used with HTML forms. Add the following to the body of your test file:

```
<?
$color = $_REQUEST["color"];
  IF ($color):
    ?>
      <p style="color: <?echo $color?>">
        You said your favorite color was <?echo $color?>.
      </p>
    <?
  ENDIF
?>
```

```

<form method="post" action="test.php">
  <p>
    What is your favorite color?
    <input type="text" name="color" value="<?echo $color?>" />
  </p>
  <input type="submit" />
</form>

```

This is a one-field form. It just asks for the reader's favorite color. The form's "action" is the file itself: any php file can also be a form interpreter. The only field we have is the field 'color', so PHP automatically creates a container called "\$color" which contains what the reader typed in that field.

The line "\$color = \$_REQUEST["color];" tells PHP to take the "request" called "color" and place that in the container called "\$color". PHP puts form data into a list called "\$_REQUEST".

You can ask for each form item by name in that list.

The line "IF (\$color):" tells PHP that we only want to do the next few lines (until the "<?ENDIF ?>") if the variable "\$color" exists and has something in it. If this is the first time the reader viewed the page, or the reader pressed the submit button without typing a favorite color, "\$color" will be empty, and those lines will be skipped.

We make use of the "echo" function to set the color of the "You said..." line, and to pre-fill the field if they've already filled out the form once.

Notice that we moved in and out of HTML in this example. We start with PHP, switch—while still in the "if" area—to HTML, and then switch back to PHP.

Sometimes you'll want to know whether the field exists rather than, or in addition to, whether it actually has anything in it. You can use a function called "isset" to determine this. Rewrite your web page so that it reads:

```

<?
  $color = $_REQUEST ["color"];
  IF (isset($color)):
  IF ($color == ""):
    echo "<p>You need to enter a color!</p>\n";
  ELSE:
  ?>
  <p style="color: <?echo $color?>">
    You said your favorite color was <?echo $color?>.
  </p>
<?

```

```

    ?>
ENDIF;
ELSE:
echo "<p>Welcome to our color extravaganza!</p>\n";
ENDIF;
<form method="post" action="test.php">
    <p>
    What is your favorite color?
    <input type="text" name="color" value="<?echo $color?>" />
    </p>
    <input type="submit" />
</form>

```

You'll notice a couple of changes here. First, we have an IF inside of our IF block. This is perfectly reasonable, and you will often do this. First, we see if the container "\$color" is "set", that is, has it been used at all. If it has, we go ahead and decide whether it has anything in it. In this case, we specifically check to see if it contains "", that is, nothing.

Here, we used two equal signs. This is the source of one of the most common mistakes in programming. When we set a container to another value, we use a single "=". When we check to see what a container contains, we use a double "==". If you use one in place of the other, you will have major problems.

If our \$color container contains nothing, we tell them they need to enter a color. Otherwise, we display their color.

Go ahead and try this script out. When you first visit the page, it should welcome you. If you try to submit the form with no color, it should tell you that you need to enter a color.

8.15 POST and GET

There are a number of "methods" that you can use to send your form data to the server for PHP to parse. Two of the most common are "POST" and "GET". These each have their own place.

The GET method is very useful if you want the viewers to be able to "come back" to the results page. For example, if you are providing a list of rooms in a building, you might use "GET" to allow them to bookmark a specific building. Or if you are providing a form that lets them search a list of classes by topic, you might use GET so that they can bookmark the topic and come back to it later to see if there are any new classes in that topic.

The GET method also allows them to copy the URL out of their web browser and send it by email.

So they could, for example, look up classes on a certain topic and then e-mail a link to those classes to a friend. Search engines often use the GET method. This allows viewers to bookmark certain searches, and it allows them to send the search results to friends or colleagues.

The POST method is not bookmarkable. You should use POST if you do not want the user to be able to “come back” to this page. For example, if they are purchasing something you don’t want them to submit the purchase twice. If they are deleting something from your database, coming back a second time will probably just result in an error; if they are inserting something into your database, they may end up inserting it twice if you aren’t careful with your PHP code.

POST information is also somewhat more secure. GET information is part of the URL. This means that it is also stored in the web server’s logs. Anyone who can see those logs can see the form information. Even if the server itself is a secure server, GET information is still posted to the logs. If your form requests secure or private information, you should use POST to submit it.

If they are going to be submitting a lot of data, you will need to use POST. Web browsers and web servers can “truncate” GET submissions. The limit on the size of a GET submission is highly variable but the general recommendation is that if the form data is likely to approach 1,024 bytes, go with the POST method. This is probably part of the reason that search engines, which want to be bookmarkable, will abbreviate their form fields to two-letter or one-letter fieldnames.

8.16 PHP with E-MAIL

There is also a function that sends e-mail from PHP. You can take form results and compile them into an e-mail message, and send that e-mail to yourself.

The ‘mail’ function has three parts: the address you’re sending to, the subject of the message, and the body of the message. Add the bold sections below:

<?

```
$color = $_REQUEST["color"];
$name = $_REQUEST["name"];
IF (isset($color)):
IF ($color == "" || $name == ""):
echo "<p>You need to enter a color and a name!</p>\n";
ELSE:
//send me an e-mail with their favorite
$subject = "$name's favorite color";
$sendto = "youraddress@wherever.com";
$message = "$name said their favorite color was $color.";
```

```

mail($sendto,$subject,$message);
//display their favorite color
$color = strtolower($color);
$color = trim($color);
IF ($color == "white"):
$bgcolor = "grey";
ELSE:
$bgcolor = "white";
ENDIF;
?>
<p style="color: <?echo $color?>; background-color: <?echo
$bgcolor?>;">
You said your favorite color was <?echo $color?>.
</p>
<?
ENDIF;
ELSE:
echo "<p>Welcome to our color extravaganza!</p>\n";
ENDIF;
?>
<form method="post" action="test.php">
<p>
What is your favorite color?
<input type="text" name="color" value="<?echo $color?>" /><br />
What is your name?
<input type="text" name="name" value="<?echo $name?>" />
</p>
<input type="submit" />
</form>

```

Here, the function (mail) takes more than one “argument”. Each item between commas, between the parentheses, is an “argument” to the function. We’re sending “mail()”arguments to specify the address the message should go to, the subject of the message, and the body of the message.

Don’t forget to replace “youraddress” with your e-mail address!

Finally, notice the two lines that begin with double slashes. PHP ignores any line that begins with double slashes. We can use this to put comments in our script. When you have any script larger than a few lines, it is very useful to comment each piece of the script so that you can remember what that piece's purpose is later.

8.17 Error Reporting

If you did not receive that error (and if PHP still did not work) it may be that you do not have error reporting turned on. At the very top of your web page, in the php, add:

```
error_reporting(E_ALL ^ E_NOTICE);
```

This turns on all errors, and then turns off notices.

Study Session Summary



Summary

In this Study Session, we discussed that

1. PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
2. PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire ecommerce sites.
3. It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
4. PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
5. PHP supports a large number of major protocols such as POP3, IMAP, and LDAP.
6. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
7. PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
8. There are four ways to escape PHP, namely Canonical PHP tags, Short-open (SGML-style) tags, ASP-style tags, HTML script tags.
9. A comment is the portion of a program that exists only for the human reader. There are two types, namely Single-line comments- used for short explanations and Multi-lines comments - to provide pseudocode algorithms and more detailed explanations when necessary.
10. A statement in PHP is any expression that is followed by a semicolon (;). Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program.
11. The main way to store information in the middle of a PHP program is by using a variable.
12. PHP has a total of eight data types which we use to construct our

-
- variables: Integers, Doubles, Booleans, NULL, Strings.
13. Arrays: are named and indexed collections of other values.
 14. Objects: are instances of programmer-defined classes
 15. A constant is a name or an identifier for a simple value.
 16. The Arithmetic Operators in PHP are: +, -, *, /, --, ++, %
 17. The Comparison Operators in PHP are: >=, <=, <, >, ==, !=
 18. The Logical (or Relational) Operators in PHP are: and, or, &&, |, !
 19. The Assignment Operators in PHP are: =, *=, /=, +=, -=, %=
 20. The Conditional (or ternary) Operator in PHP is ?:
 21. The operator precedence in PHP can be represented by an acronym: UMARELLCA (U-Unary, M-Multiplication, A-Addition, Relational, C-Comparison, L-Logical, C-Conditional, A-Assignment).
 22. There are two methods that can use to send form data to the server for PHP to parse. They are “POST” and “GET”. This each has its own place.
 23. PHP has feature for capturing error.

Assessment



Assignment

1. What are variables, data types and constants?
2. Mention the types of operators you know, with examples
3. What are functions?
4. Why do we need to use functions?
5. Write a PHP Script to compute the age of a student who enters his/her date of birth.
6. What is a Statement?
7. Differentiate between a Variable and a Constant.

Study Session 9

Concurrency Programming for the Web

Expected duration: 1 week or 2 contact hour

Introduction

In the previous study session, we learnt how to use Hypertext Pre-processor (PHP) to create dynamic webpages by adding various functions like; sending email, sharing data from page to page. e.t.c to our website. In this study session, we will discuss how an operating system manage multiple requests by the same user without having to have multiple copies of the programming running in the computer.

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 9.1 use concurrency
- 9.2 use thread
- 9.3 express multithreading
- 9.4 create threads

Terminology

| | |
|-----------------------|--|
| Concurrency | The ability of a database to allow multiple users to affect multiple transactions. |
| Multithreading | The ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer. |
| Multiprocessor | The simultaneous execution of two or more programs or instruction sequences by separate CPUs under integrated control. |

9.1 Concurrency

Concurrency

The property of program, algorithm, or problem decomposability into order-independent or partially-ordered components or units.

Multiprocessor

A computer with more than one central processor.

If you have a **multiprocessor** or a distributed system you will have **concurrency**, since in these systems there is more than one CPU executing instructions. Similarly, most hardware has separate parts that can change state simultaneously and independently. But suppose your system consists of a single CPU running a program. Then you can certainly arrange for concurrency by multiplexing that CPU among several tasks, but why would you want to do this? Since the CPU can only execute one instruction at a time, it isn't entirely obvious that there is any advantage to concurrency. Why not get one task done before moving on to the next one?

There are only two possible reasons:

1. A task might have to wait for something else to complete before it can proceed, for instance for a disk read. But this means that there is some concurrent task that is going to complete, in the example an I/O device, the disk. So we have concurrency in any system that has I/O, even when there is only one CPU.
2. Something else might have to wait for the result of one task but not for the rest of the computation, for example a human user. But this means that there is some concurrent task that is waiting, in the example the user. Again we have concurrency in any system that has I/O.

In the first case one task must wait for I/O, and we can get more work done by running another task on the CPU, rather than letting it idle during the wait. Thus the concurrency of the I/O system leads to concurrency on the CPU. If the I/O wait is explicit in the program, the programmer can know when other tasks might run; this is often called a 'non-preemptive' system, because it has sequential semantics except when the program explicitly allows concurrent activity by waiting. But if the I/O is done at some low level of abstraction, higher levels may be quite unaware of it. The most insidious example of this is I/O caused by the virtual memory system: every instruction can cause a disk read. Such a system is called 'preemptive'; for practical purposes a task can lose the CPU at any point, since it's too hard to predict which memory references might cause page faults.

In the second case we have a motivation for true preemption: we want some tasks to have higher priority for the CPU than others. An important special case is interrupts, discussed below.

A concurrent program is harder to write than a sequential program, since there are many more possible paths of execution and interactions among the parts of the program. The canonical example is two concurrent executions of

```
x := x + 1
```

Since this command is not atomic, x can end up with either 1 or 2, depending on the order of execution of the expression evaluations and the

$x := x + 1$ assignments. The interleaved order

evaluate $x + 1$

evaluate $x + 1$

$x := \text{result}$

$x := \text{result}$

leaves $x = 1$, while doing both steps of one command before either step of the other leaves

$x = 2$. This is called a race, because the two threads are racing each other to get x updated.

Apart from recent hardware trends towards multi-core and multiprocessor systems, the use of concurrency in applications is generally motivated by performance gains. There are three different fundamental ways on how the concurrent execution of an application can improve its performance:

9.2 Reduce Latency

A unit of work is executed in shorter time by subdivision into parts that can be executed concurrently.

9.2.1 Hide latency

Multiple long-running tasks are executed together by the underlying system. This is particularly effective when the tasks are blocked because of external resources they must wait upon, such as disk or network I/O operations.

9.2.2 Increase throughput

By executing multiple tasks concurrently, the general system throughput can be increased. It is important to notice that this also speeds up independent sequential tasks that have not been specifically designed for concurrency yet. The presence of concurrency is an intrinsic property for any kind of distributed system. Processes running on different machines form a common system that executes code on multiple machines at the same time.

Conceptually, all web applications can be used by various users at the same time. Thus, a web application is also inherently concurrent. This is not limited to the web server that must handle multiple client connections in parallel. Also the application that executes the associated business logic of a request and the backend data storage are confronted with concurrency.

9.3 Multithreading

Multithreading

A technique by which a single set of code can be used by several processors at different stages of execution.

Threads are lightweight processes; they share the same address space. In Multithreaded environment, programs make maximum use of CPU so that the idle time can be kept to minimum. The main purpose of **multithreading** is to provide simultaneous execution of two or more parts of a program to maximum utilize the CPU time. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program called a thread. Each thread has a separate path of its execution. So this way a single program can perform two or more tasks simultaneously.

There are several thread states, A thread can be in any one of the state at a particular point of time. It can be running state. It can be ready to run state as soon as it gets CPU time. A running thread can be suspended. A suspended thread can be resumed. A thread can be blocked when waiting for a resource. At any time a thread can be terminated.

Handling of multithreading in java is quite simple. A thread can be created in two ways: 1) By extending Thread class 2) By implementing Runnable interface.

9.3.1 Method 1: Thread creation by implementing Runnable Interface

- One way of creating a thread is to create a class that implements the Runnable interface. We must need to give the definition of run() method.
- This run method is the entry point for the thread and thread will be alive till run method finishes its execution.
- Once the thread is created it will start running when start() method gets called. Basically start() method calls run() method implicitly.

A Simple Example

```
class MultithreadingDemo implements Runnable{
    public void run(){
        System.out.println("My thread is in running state.");
    }
    public static void main(String args[]){
        MultithreadingDemo obj=new MultithreadingDemo();
        Thread tobj =new Thread(obj);
        tobj.start();
    }
}
```

```
}
```

Output:

My thread is in running state.

9.3.2 Method 2: Thread creation by extending Thread class

This is the second way of creating a thread. Here we need to create a new class that extends the Thread class.

- The class should override the run() method which is the entry point for the new thread as described above.
- Call start() method to start the execution of a thread.

We will explain this method by using the same above examples. Example 1:

```
class MultithreadingDemo extends Thread{
    public void run(){
        System.out.println("My thread is in running state.");
    }
    public static void main(String args[]){
        MultithreadingDemo obj=new MultithreadingDemo();
        obj.start();
    }
}
```

Output:

My thread is in running state.

Study Session Summary



Summary

In this Study Session, you learnt that

1. A web application is also inherently concurrent because all web applications can be used by various users at the same time
2. Threads are lightweight processes.
3. There are several thread states: running state, ready state, suspended state, resumed, blocked, waiting, terminated.
4. A thread can be created in two ways:
 - a) By extending Thread class

b) By implementing Runnable interface.

Assessment



Assignment

1. Explain the term Concurrency?
2. What do you understand by thread
3. Give reason(s) for implementing multithreading.

Study Session 10

Website Maintenance

Expected duration: 1 week or 2 contact hour

Introduction

In the previous study session, we explained concurrency as the ability of a database to allow multiple users to affect multiple transactions. In the study session, we will discuss how to maintain a website, the team responsible for website maintenance.

Learning Outcomes



Outcomes

When you have studied this session, you should be able to:

- 10.1 *define* website maintenance
- 10.2 *state* the activities involved in website maintenance
- 10.3 *identify* five website maintenance team members and what they do.

Terminology

| | |
|----------------------------|---|
| Website Maintenance | Keeping the current website content updated and accurate. It is not meant to be used to completely redesign or reinvent your website. |
|----------------------------|---|

10.1 Website Maintenance Team

Website Maintenance

Keeping the current website content updated and accurate. It is not meant to be used to completely redesign or reinvent your website.

A **Website Maintenance** Team is responsible for expediting the tasks of site maintenance.

The variety of roles on such a team is usually quite broad and may include the following:

- Website Maintenance Team Leader
- Publishing Representative (Editor)
- Quality Assurance Representative
- Feedback Monitoring Representative
- Website Performance Representative
- Infrastructure Monitoring Representative

-
- Change Control Representative

It is worth bearing in mind that if your site is small, you may not have to allocate one person for each activity. Instead, you could combine several roles together.

For example, an Editor could also act as a Team Leader, as well as look after Quality Assurance or Feedback Monitoring.

Yet, even such small teams must still have all necessary skills represented. This is because each one is vital for maintaining a quality web presence.

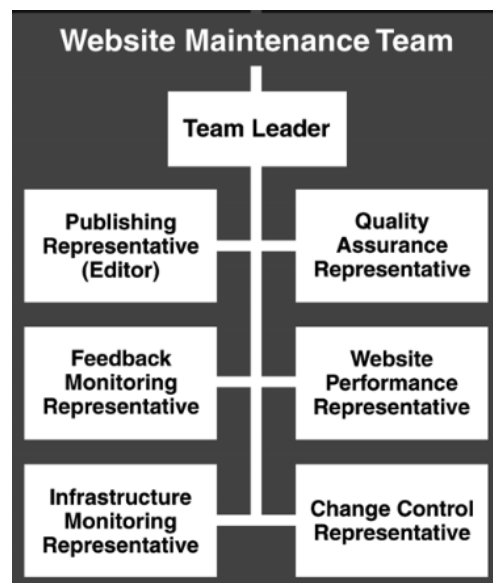


Figure 10.1: Website maintenance team

10.2 Website Scale

Website Scale is a means of classifying a site in terms of three parameters:

- Size
- Complexity
- Levels of activity

Any site can be represented in this way—from a small, plain text website to a massive corporate intranet.

10.3 Website Size

A website's size is an estimate of the total man-hours required to produce and maintain all the content that it contains. This can then be used to calculate the number of people required for support—particularly the activities of Website Publishing and Quality Assurance.

10.4 Website Complexity

Complexity is a function of the intricacy of the technology used to develop a site. There are three levels of website complexity:

10.4.1 Basic Website

Sites that simply contain plain text with perhaps a few supporting images and downloads, e.g. PDFs.

10.4.2 Dynamic Website

On a Dynamic site content is stored in a database and published according to the requirements of site visitors. Some also offer basic interactive services, e.g. Discussion Forums.

10.4.3 Transactional Website

A Transactional website is one that uses the internet for facilitating business operations or generating revenue. Sites of this type rely on databases and other advanced technology for collecting and processing orders. Some indicative figures for the staffing of a Technical Team are indicated in the following table:

10.5 Website Activity

Website Activity is a measure of the traffic experienced by a site, e.g. Page Impressions, Visitors, Visits, etc. A website with heavy activity is unlikely to function properly without a full complement of maintenance personnel.

A Busy site that is also Large in size and Transactional in nature may need dozens of staff to keep it going.

The three most frequently monitored figures include:

- Visits: A visit is an instance of a unique visitor accessing a website.
- Visitors: A visitor is the originator of a visit, i.e. the person who browses a website.
- Page Impressions: A Page Impression is a 'hit' on a page that contains content. (The recent growth of Rich Internet Applications built on AJAX and Flash means there is now less emphasis on this as a metric of success.)

10.6 Regular Website Maintenance Tasks

There are some regular website maintenance tasks you should perform on a scheduled basis. Scheduling at least monthly would be the timeline to start with.

10.6.1 Backing up Website

Backing up your website is something you should do all the time, especially if you are the type that uses the online interface of your store or blog to make changes. Things happen. Even though the web hosting company says they backup the sites on their servers, their last backup could have been before your last edit. If the server crashes for some reason or your site gets hacked, your edits will be gone if the web hosting company restores what they had backed up. Imagine losing a whole day's work, just because you didn't take a few minutes to backup the site.

10.6.2 Monitor Website Outages

If your site goes down, you want to be the first to know and not receive an email from someone else they cannot access your site.

SiteUp is a small program that runs on your computer in the background checking your site on a regular basis. It will notify you when the site is down with a popup. Obviously though, your computer has to be on for it work.

10.6.3 Check Domain Registration Information

Look up in the WHOIS records what information is recorded for your domain name. Make sure it is correct. Sometimes when you initially sign up for your domain you would have used an email address that is no longer valid. This needs to be updated as when there is a problem with your domain or an expiry notice is sent out you won't get the emails. They are sent to the email address on record.

10.6.4 Test Website Speed

Testing the download speed of your site regularly is important, especially if you have added a new feature. Web surfer has a very short attention span. If your site is slow to load, they are not going to wait. You need to do everything you can to improve the download speed of your site or blog so visitors stay to read your content and hopefully provide you with organic incoming links by spreading the word for you what a wonderful site you have.

10.6.5 Link Check

Links become broken over time. With changes within the site and if you referenced someone in one of your articles or somewhere else within the site links could have changed or are broken.

The task to find broken links isn't too hard. Just use a link checker to test your external links and internal links at least once a month.

10.6.6 Software Updates

Third party software, like your ecommerce software, WordPress and Joomla for example, are always updating their software. You need to

keep on top these updates and install them as soon as they come out. The updates won't just be new features, they will include security updates too.

10.6.7 Analyze Stats

Analyze not just your sales stats but your website stats too

10.6.8 Traffic Stats

Look at your web server stats to determine your website traffic. If your web hosting account doesn't have website stats then get one installed. Something like Awstats that provides:

- Pages entered on and left on
- Time spent on the site
- Bounce rate
- Referring sites
- Countries your visitors are from
- Keywords/phrase that were used to find you

Google Analytics will provide some of this information. It may not be as complete as a website stats program that is run from your actual server. Are you showing up on the first page for the keywords/phrase you want to? If you have given it some time, e.g. a few months, to get onto the first page of the search results naturally then maybe it is time to look at your content and revise it.

One thing a website stats program installed on your server will do that Google Analytics doesn't is show you who is hotlinking (linking directly to your images on your site). e.g. your images, PDFs, reports, etc.. These people are stealing your content and your bandwidth if they do not have your permission to do so. With this information you can stop the hotlink.

Are you showing up on the first page for the keywords/phrase you want to? If you have given it some time, e.g. a few months, to get onto the first page of the search results naturally then maybe it is time to look at your content and revise it.

10.6.9 Reputation Management

Using Google Alerts, you can monitor your website name, your name, your brand and your content on the web. You will know who is talking about you. This gives you an opportunity to jump into the conversation. Thank those who are praising you. Fix a problem that is being discussed related to your business.

Tracking your website address with Google Alerts is 2 fold.

1. You see who is linking to you and can pop over there and say thanks.
2. You can catch the use of your content without your permission.

10.7 Development Server or Live Server

If you have a version of your site already live and you are working on an update, then you will need to have a different place for the new site to be

tested — this could be as simple as a different folder on your computer or it could be a separate server altogether.

If you are making changes to an existing site, you should always work on a separate copy of the site rather than the version the public will be looking at. It is also good practice to have a backup copy of each version of the site.

Once you have finished the following tests, you can then make the site live, ready for the public.

Study Session Summary



Summary

In this Study Session, we learnt that;

1. Website Maintenance comprises all the activities needed to ensure the operational integrity of your website.
2. The following are some of the website maintenance team: Website Maintenance Team Leader, Publishing Representative (Editor), Quality Assurance Representative, Feedback Monitoring Representative, Website Performance Representative, Infrastructure Monitoring Representative, Change Control Representative.

Assessment



Assignment

1. What do you understand by website maintenance?
2. What are some of the activities involved in website maintenance?
3. Name five website maintenance team members and what they do.

References

- Basic CSS (2008) Dwight VanTuyl. The LINGUIST List
- Castro, E. (2003). HTML for the World Wide Web, Fifth Edition, with XHTML and CSS: Visual QuickStart Guide. 14
- Concurrent Programming for Scalable Web Architectures. Institute of Distributed Systems Faculty of Engineering and Computer Science, Ulm University. VS-D01-2012.
- Corey Benson & Robert Girardin. A Guide to Understanding Web Application Development, SAS Institute, Inc., Cary, NC, SAS Institute, Inc., Cary, NC
- Emmanuel Benoist.(2013-14) Web Programming in Javascript. Fall Term 2013-14
- Introduction to HTML/XHTML Handout Companion to the Interactive Media Center's Online Tutorial. Interactive Media Center <http://library.albany.edu/imc/> 518 442-3608.
- Jon Duckett (2008) Beginning Web Programming with HTML, XHTML, and CSS. Published by Wiley Publishing, Inc. 10475 Crosspoint Boulevard Indianapolis, IN 46256 www.wiley.com. Copyright © 2008 by Wiley Publishing, Inc., Indianapolis, Indiana. ISBN: 978-0-470-25931-3.
- Paul De Bra, Natalia Stash and David Smits. Creating Adaptive Web-Based Applications. Eindhoven University of Technology. Department of Computer Science. PO Box 513, NL 5600 MB Eindhoven, Netherlands
- Php for dynamic web pages. (2008) by Jerry Stratton
- Php Tutorial Point: Simply Easy Learning. © Copyright 2015 by Tutorials Point (I) Pvt. Ltd.
- Practical Concurrency 6.826—Principles of Computer Systems
- Sabah Al-Fedaghi (2011) Developing Web Applications. International Journal of Software Engineering and Its Applications Vol. 5 No. 2, April, 2011
- Shane Diffily Website Maintenance: An Abridged extract from the Website Manager's handbook by Gerry McGovern.
- <http://www.answers.com/topic/computer-1>
- <http://www.webopedia.com>
- <http://www.whatis.com>
- <http://www.ianswer4u.com/2012/01/tree-topology-advantages-and.html#axzz3enldQQxn>
- <http://www.buzzle.com/articles/advantages-and-disadvantages-of-different-network-topologies.html>

<http://www.journaldev.com/1854/java-web-application-tutorial-for-beginners> Access date: 5th September, 2015

<http://www.vogella.com/tutorials/JavaWebTerminology/article.html>

Access date: 5th September, 2015

http://www.sqa.org.uk/e-learning/ClientSide01CD/page_18.htm

Access date: 6th September, 2015

<https://www.techopedia.com/definition/23898/web-programming>

Access date: 6th September, 2015

<http://www.webpagemistakes.ca/maintain-website/>