IBA CAMPUS

BMI CAMPUS

AeU asiaeuniversity

Bachelor of Information Communication Technology (Hons)

Diploma in Information Communication Technology

Academic Year I

Module Name:

WEB APPLICATION DEVELOPMENT

Module Outline:

1. Introduction to Internet, WWW and Web Browsers
2. Introduction to What is a Website
3. Introduction to Web Application
4. Introduction to Web Development
5. Introduction to HTML
6. Introduction to CSS
7. Introduction to PHP

01. Introduction to Internet, WWW and Web Browsers

Internet is a global communication system that links together thousands of individual networks. It allows exchange of information between two or more computers on a network. Thus internet helps in transfer of messages through mail, chat, video & audio conference, etc. It has become mandatory for day-to-day activities: bills payment, online shopping and surfing, tutoring, working, communicating with peers, etc.

In this topic, we are going to discuss in detail about concepts like basics of computer networks, concept of internet, basics of internet architecture, services on internet, World Wide Web and websites, communication on internet, internet services, preparing computer for internet access, ISPs and internet access techniques, web browsing software, popular web browsing software, configuring web browser, search engines, popular search engines/search for content, accessing web browser

No	Internet, WWW, Web Browsers Concepts & Description
1	Basics of Computer Networks Computer network is an interconnection between two or more hosts/computers. Different types of networks include LAN, WAN, MAN, etc.
2	Internet Architecture Internet is called the network of networks. It is a global communication system that links together thousands of individual networks. Internet architecture is a meta-network, which refers to a congregation of thousands of distinct networks interacting with a common protocol
3	Services on Internet Internet acts as a carrier for numerous diverse services, each with its own distinctive features and purposes.
4	Communication on Internet communication can happens through the the Internet by using Email, Internet Relay Chat, Video Conference etc.
5	Web Browsing Software "World Wide Web" or simple "Web" is the name given to all the resources of internet. The special software or application program with which you can access web is called "Web Browser".
6	Configuring Web Browser Search Engine is an application that allows you to search for content on the web. It displays multiple web pages based on the content or a word you have typed.
7	Search Engines Search Engine is an application that allows you to search for content on the web. It displays multiple web pages based on the content or a word you have typed.
8	Search for the content Search Engine helps to search for content on web using the different stages
9	Accessing Web Browser There are several ways to access a web page like using URLs, hyperlinks, using navigating tools, search engine, etc.

Computer Concepts - Services on Internet

Internet acts as a carrier for numerous diverse services, each with its own distinctive features and purposes.

World Wide Web and Websites

- World Wide Web is being used on internet right now. WWW is the name given to all resources of the internet, which you can access with a web browser. It was created as a method for incorporating footnotes, figures and cross-references into online documents in the European Particle Physics Laboratory in Geneva, Switzerland in 1989. The web makers wanted to make a simple method to access documents that are stored on a network, without searching through indexes or directories of files, and without physically copying documents from one computer to another before viewing them. To do this, they made a way to "connect" documents that were stored in different locations on a single computer, or different computers on a network.

Terminologies related to WWW

- Web documents can be linked together, and are called "Hypertext". Hypertext systems offer an easy approach to manage huge collections of data, which includes text files, pictures, sounds, movies and more. In a hypertext system, when you view a document or your computer screen, you can also access all the data that is linked to it. To support hypertext documents, web uses a protocol called "Hypertext Transfer Protocol" (HTTP). A hypertext document is a specially encoded file that uses "Hypertext Markup Language" (HTML). HTTP and Links are foundation for WWW. Web page is displayed in the web browser. It is a kind of word processing document which contains pictures, sounds and even movies along with text.

Computer Concepts - Web Browsing Software

"World Wide Web" or simple "Web" is the name given to all the resources of internet. The special software or application program with which you can access web is called "Web Browser".

There's an entire history of web browsers. Before the web browsers we knew today,

- 1990 – The WorldWideWeb (not to be confused with the World Wide Web) was the first browser ever created by W3C Director Tim Berners-Lee, then renamed Nexus to differentiate from the actual World Wide Web. Unlike today, this was the only browser and the only way to access the web.
- 1992 – Lynx was a texted-based browser that couldn't display any graphic content.

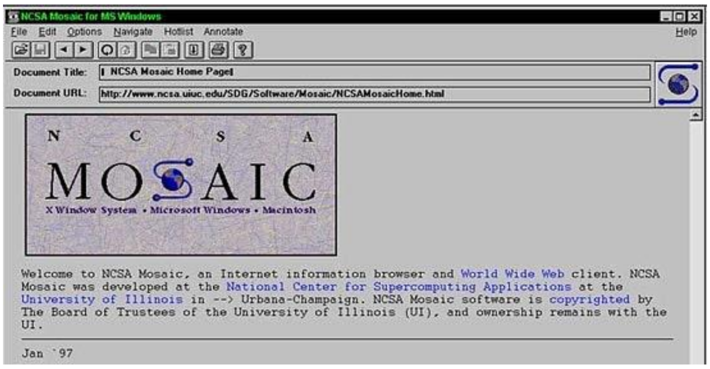
- 1993 – Mosaic was the first browser to allow images embedded in text making it “the world’s first most popular browser”.
- 1994 – A noticeable improvement to Mosaic came Netscape Navigator.
- 1995 – Internet Explorer made its debut as Microsoft’s first web browser.
- 1996 – Opera started as a research project in 1994 that finally went public two years later. This was also arguably the beginning of the browser wars, mainly between IE 3 and Navigator 3 as Internet Explorer inched ahead with new capabilities.
- 2003 – Apple’s Safari browser was released specifically for Macintosh computers instead of Navigator.
- 2004 – Mozilla launched Firefox as Netscape Navigator faded out.
- 2007 – Mobile Safari was introduced as Apple’s mobile web browser and continues to dominate the iOS market.
- 2008 – Google Chrome appeared to soon take over the browser market.
- 2011 – Opera Mini was released to focus on the fast-growing mobile browser market.
- 2015 – Microsoft Edge was born to combat Google.

List of Top 09 Internet Browsers - PC

1. Chrome
2. Firefox
3. IE
4. Safari
5. Edge
6. Opera
7. UC Browser
8. Brave
9. Others – Chromium, Maxthon

List of Top 10 Internet Browsers – Mobile

1. Chrome
2. Safari
3. UC Browser
4. Samsung Internet
5. Opera
6. Android
7. KaiOS

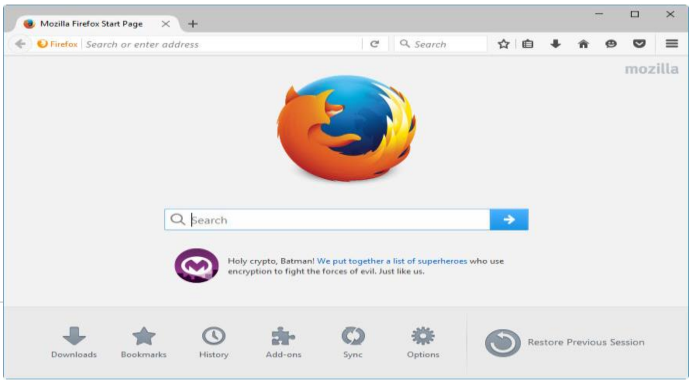

NCSA Mosaic for MS Windows
File Edit Options Navigate Hotlist Annotate Help
NCSA MOSAIC
X Window System • Microsoft Windows • Macintosh

Document Title: NCSA Mosaic Home Page

Document URL: http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html

Welcome to NCSA Mosaic, an Internet information browser and World Wide Web client. NCSA Mosaic was developed at the National Center for Supercomputing Applications at the University of Illinois in --> Urbana-Champaign. NCSA Mosaic software is copyrighted by The Board of Trustees of the University of Illinois (UI), and ownership remains with the UI.

Jan `97



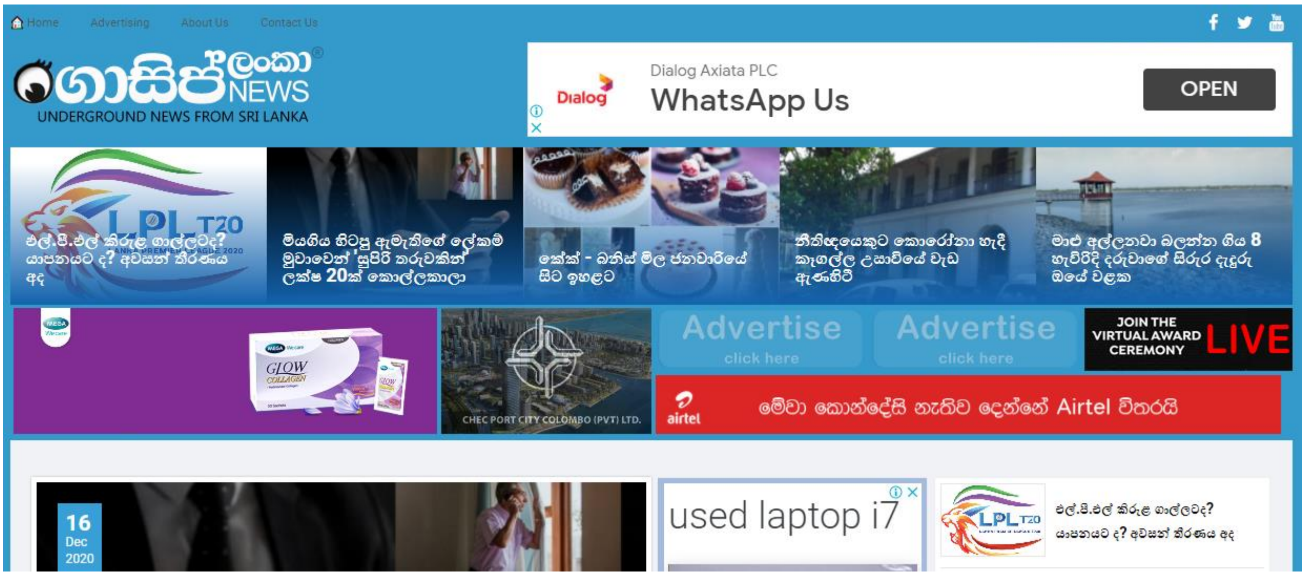
Holy crypto, Batman! We put together a list of superheroes who use encryption to fight the forces of evil. Just like us.

8. QQ Browser
9. Firefox
10. Other – Android Native Browser, KaiOS, and QQ Browser

What is a Website?

A website is a group of globally accessible, interlinked web pages which have a single domain name. It can be developed and maintained by an individual, business or organization. The website aims to serve a variety of purposes. Example: Blogs.

A website is hosted on a single or multiple web server. It is accessible via a network like the Internet or a private local area network via IP address.



එල්.පී.එල් කිරුළ ගාල්ලටද? යාපනයට ද? අවසන් තීරණය අද

මියගිය හිටපු ඇමැතිගේ ලේකම් මුවාවෙන් 'සුපිරි තරුවකින්' ලක්ෂ 20ක් කොල්ලකාලා

කේක් - බනිස් මිල ජනවාරියේ සිට ඉහළට

නීතිඥයෙකුට කොරෝනා හැදී කෑගල්ල උසාවියේ වැඩ ඇණහිටී

මාළු අල්ලනවා බලන්න ගිය 8 හැවිරිදි දරුවාගේ සිරුර දැදුරු ඔයේ වළක

මේවා කොන්දේසි නැතිව දෙන්නේ Airtel විතරයි

එල්.පී.එල් කිරුළ ගාල්ලටද? යාපනයට ද? අවසන් තීරණය අද

Why you need a Website?

Here, are prime reasons why someone need a website:

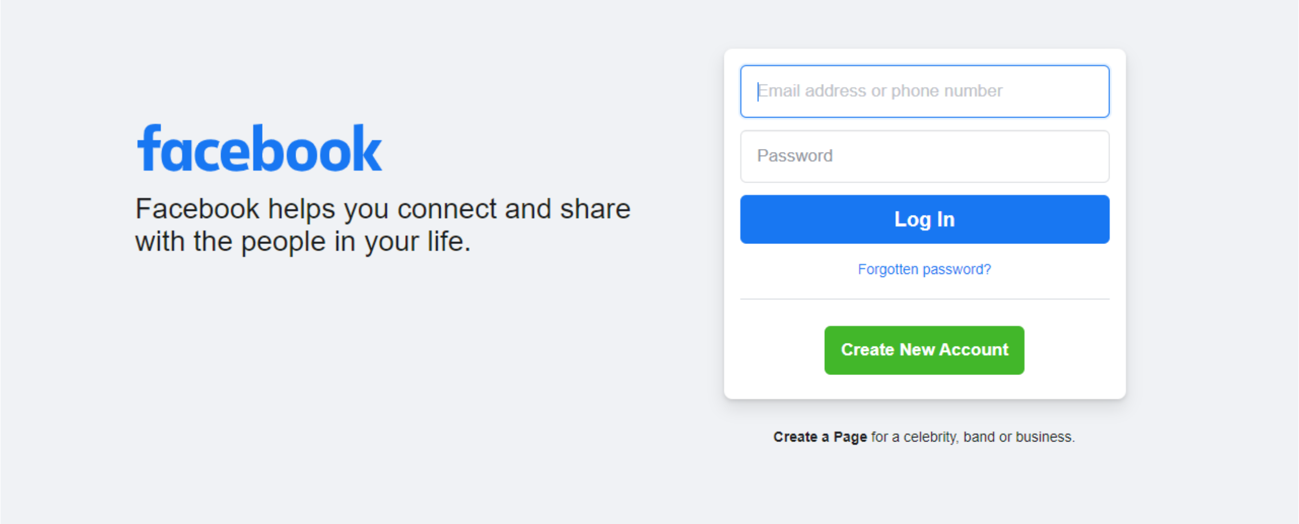
- An effective method to showcase your products and services
- Developing a site helps you to create your social proof
- Helps you in branding your business
- Helps you to achieve your business goals
- Allows you to increase your customer support

Characteristics Of Website

- Quality and relevant Web Content is which richly displayed.
- User-friendly navigation and web design
- Can be easily searched using search engines like Google.

What is a Web Application?

A web application is a software or program which is accessible using any web browser. Its frontend is usually created using languages like HTML, CSS, Javascript, which are supported by major browsers. While the backend could use any programming stack like LAMP, MEAN, etc. & programming languages PHP, Python, Ruby, Java, C#. Unlike mobile apps, there is no specific SDK (Software development kit) developing web applications.


facebook
Facebook helps you connect and share with the people in your life.
Email address or phone number
Password
Log In
Forgotten password?
Create New Account
Create a Page for a celebrity, band or business.

Web applications are more popular because of the following reasons

- Compared to desktop applications, web applications are easier to maintain by as they use the same code in the entire application. There are no compatibility issues.
- Web applications can be used on any platform: Windows, Linux, Mac… as they all support modern browsers.
- Mobile App store approval not required in web applications.
- Released any time and in any form. No need to remind users to update their applications.
- You can access these web applications 24 hours of the day and 365 days a year from any PC.
- You can either make use of the computer or your mobile device to access the required data.
- Web applications are a cost-effective option for any organization. Seat Licenses for Desktop software are expensive.
- Web-Based Apps are Internet-enabled apps that are accessed through the mobile's web browser. Therefore, you don't require to download or install them.

Below given are the prime difference between web application and web site:

Parameter	Web Application	Website
Created for	A web application is designed for interaction with the end user	A website mostly consists of static content. It is publicly accessible to all the visitors.

User interaction	In a web application, the user not only read the page content but also manipulate the restricted data.	A website provides visual & text content which user can view and read, but not affect it 's functioning.
Authentication	Web applications need authentication, as they offer a much broader scope of options than websites.	Authentication is not obligatory for informational websites. The user may ask to register to get a regular update or to access additional options. This features not available for the unregistered website visitors.
Task and Complexity	Web application functions are quite higher and complex compared to a website.	The website displays the collected data and information on a specific page.
Type of software	The web application development is part of the website. It is itself not a complete website.	The website is a complete product, which you access with the help of your browser.
Compilation	The site must be precompiled before deployment	The site doesn't need to be pre-compiled
Deployment	All changes require the entire project to be re-compiled and deployed.	Small changes never require a full re-compilation and deployment. You just need to update the HTML code.

What is Web Development?

Web development is basically the tasks associated with developing websites for hosting via intranet or internet. The web development process involves web design, web content development, client-side/server-side scripting and network security configuration.

A website can either be a simple one-page site, or it could be an incredibly complex web application. When you view your website on the web in a browser, it is because of all the processes involved in web development.

The web development hierarchy is as follows:

1. Client-side coding (Front End).
2. Server-side coding (Back End).
3. Data and Databases.

Basics: Web Development What we need?

Websites are a bunch of files stored on computers called **servers**. The Servers are computers that are used to host websites and store the website files. These servers are connected to the giant network called the **World Wide Web**.

Browsers are programs that you run on your computer. They load the website files via your internet connection.

01.Client-side coding (Front End)

there are 3 main components that make up every website:

HTML – Hyper Text Markup Language (HTML)

- Is the foundation of all websites It's the main file type that is loaded in your browser when you look at a website. This scripting language is used to structure the different parts of our content and define what their meaning or purpose is.

CSS – Cascading Style Sheets (CSS)

- is used for styling the HTML elements. It provides 1000s of styling functions which are used to style the HTML elements defined by us. It is the language that we use to style and layout our web content.

JavaScript

- This programming language allows you to interact with elements on the website and to manipulate them. While CSS adds style to HTML, JavaScript adds interactivity and makes a website more dynamic.

Front End Skills

It is important to make sure that web applications download fast and are responsive to user interaction, regardless of a users bandwidth, screen size, network, or device capabilities. The intermediate Front End Skills include:

Responsive Design

- We use different gadgets like computers, phones, and tablets to look at web pages. The web pages adjust themselves to the device you're using without any extra effort from your end. This is due to the responsive design. One major role of a front end developer is to understand the responsive design principles and how to implement them on the coding side. It is an intrinsic part of CSS frameworks like the Bootstrap. These skills are all interconnected and so as you learn one you'll often be making progress in the others at the same time.

Build Tools

- The modern web browsers come equipped with developer tools for testing and debugging. These tools allow you to test the web pages in the browser itself and finds out how the page is interpreting the code. Browser developer tools usually consist of an inspector and a JavaScript console. The inspector allows you to see what the runtime HTML on your page looks like, what CSS is associated with each element on the page. The JS console allows you to view any errors that occur as the browser tries to execute your JS code.

Version Control / Git

- Version control is the process of tracking and controlling changes to your source code so that you don't have to start from the beginning if anything goes wrong. It is a tool that is used to track the changes made previously so that you can go back to a previous version of your work and find out what went wrong without tearing the whole thing down.

02. Server-side coding (Back End)

The back-end layer forms a dynamic connection between the front-end and the database. To get this layer working it's important to know at least one of the programming languages such as Python, Java, PHP, Ruby, etc and knowledge of server-side frameworks such as NodeJS is mandatory.

Python

- is an open-source, object-oriented programming language, one of the favorite languages of most software and web developers.

Java

- is an open-source, high-level programming language which was released by Sun Microsystems in 1996. It follows the Write Once Run Anywhere (WORA) approach that makes it compatible to run on any platform.

PHP

- is an open-source, server-side scripting language used to develop the back-end logic of an application. It is a powerful tool for making dynamic and interactive websites.

Ruby

- A dynamic, open source, object orinted programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write

Perl

- Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more

NodeJS

- is an open-source, JavaScript framework used specifically for creating the back end or the server-side of an application. Through NodeJS, JavaScript can now finally run on the server-side of the web.

03. Data and Database

The data layer is a massive warehouse of information. It contains a database repository which captures and stores information from the front-end, through the back-end. A prerequisite is to have knowledge of how data is stored, edited, retrieved, etc. An understanding of Databases such as MySQL, MongoDB is a must.

MySQL is an open-source, Relational Database Management System that provides multi-user access and supports multi storage engines.

MongoDB is known for its ease of use and its quickness in handling a large amount of data. It is an open-source, object-oriented, NoSQL database which is highly scalable and is efficient in handling unstructured data.

Server and Deployment

Servers are basically computers that store website files and other resources like databases.

Server Setup

To make a website accessible publicly on the internet, it needs to be installed on a server. Once you have your domain name and server space, it’s time to set up the site on the server. The first thing is to direct the domain name to the server’s unique IP address. Then you need to set up website files and finally the database and other configurations.

02. Introduction to HTML

- HTML stands for Hyper Text Markup Language
- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

HTML was created by Sir Tim Berners-Lee in late 1991. HTML is a very evolving markup language and has evolved with various versions updating. Long before its revised standards and specifications are carried in, each version has allowed its user to create web pages in a much easier and prettier way and make sites very efficient.

- HTML 1.0 was released in 1993 with the intention of sharing information that can be readable and accessible via web browsers. But not many of the developers were involved in creating websites. So the language was also not growing.
- Then comes the HTML 2.0, published in 1995, which contains all the features of HTML 1.0 along with that few additional features, which remained as the standard markup language for designing and creating websites until January 1997 and refined various core features of HTML.
- Then comes the HTML 3.0, where Dave Raggett who introduced a fresh paper or draft on HTML. It included improved new features of HTML, giving more powerful characteristics for webmasters in designing web pages. But these powerful features of new HTML slowed down the browser in applying further improvements.

- Then comes HTML 4.01, which is widely used and was a successful version of HTML before HTML 5.0, which is currently released and used worldwide. HTML 5 can be said for an extended version of HTML 4.01, which was published in the year 2012.

A Simple HTML Document

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
</head>
<body>
    <h1>My First Heading</h1>
    <p>My first paragraph.</p>
</body>
</html>
```

Example Explained

- The <!DOCTYPE html> declaration defines that this document is an HTML5 document
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the HTML page
- The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The <h1> element defines a large heading
- The <p> element defines a paragraph

What is an HTML Element?

An HTML element is defined by a start tag, some content, and an end tag

```
<tagname>Content goes here...</tagname>
```

The HTML element is everything from the start tag to the end tag

```
<h1>My First Heading</h1>
<p>My first paragraph.</p>
```

Note: Some HTML elements have no content (like the <br> element). These elements are called empty elements. Empty elements do not have an end tag!

What is an HTML Attributes

- All HTML elements can have attributes
- Attributes provide additional information about elements
- Attributes are always specified in the start tag
- Attributes usually come in name/value pairs like: name="value"

Example : The href Attribute

The <a> tag defines a hyperlink. The **href** attribute specifies the URL of the page the link goes to

Correct way

```
<a href="https://www.w3schools.com/html/">HTML Tutorial</a>
```

Wrong way

```
<a href=https://www.w3schools.com/html/>HTML Tutorial</a>
```

Chapter Summary

- All HTML elements can have **attributes**
- The href attribute of <a> specifies the URL of the page the link goes to
- The src attribute of <img> specifies the path to the image to be displayed
- The width and height attributes of <img> provide size information for images
- The alt attribute of <img> provides an alternate text for an image
- The style attribute is used to add styles to an element, such as color, font, size, and more
- The lang attribute of the <html> tag declares the language of the Web page
- The title attribute defines some extra information about an element

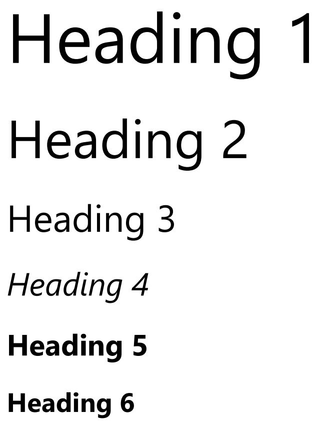
HTML Headings

HTML headings are titles or subtitles that you want to display on a webpage.

HTML headings are defined with the <h1> to <h6> tags.

<h1> defines the most important heading. <h6> defines the least important heading.

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```



Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

HTML Paragraphs

The HTML <p> element defines a paragraph.

A paragraph always starts on a new line, and browsers automatically add some white space (a margin) before and after a paragraph.

A paragraph always starts on a new line, and is usually a block of text.

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

You cannot be sure how HTML will be displayed.

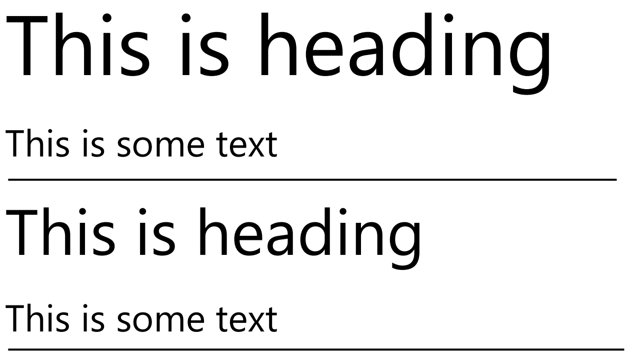
Large or small screens, and resized windows will create different results.

With HTML, you cannot change the display by adding extra spaces or extra lines in your HTML code. The browser will automatically remove any extra spaces and lines when the page is displayed.

HTML Horizontal Rules

The <hr> tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule. The <hr> element is used to separate content (or define a change) in an HTML page

```
<h1>This is heading 1</h1>
<p>This is some text.</p>
<hr>
<h2>This is heading 2</h2>
<p>This is some other text.</p>
<hr>
```



This is heading

This is some text

This is heading

This is some text

HTML <img> Tag

The <img> tag is used to embed an image in an HTML page.

Images are not technically inserted into a web page; images are linked to web pages. The <img> tag creates a holding space for the referenced image.

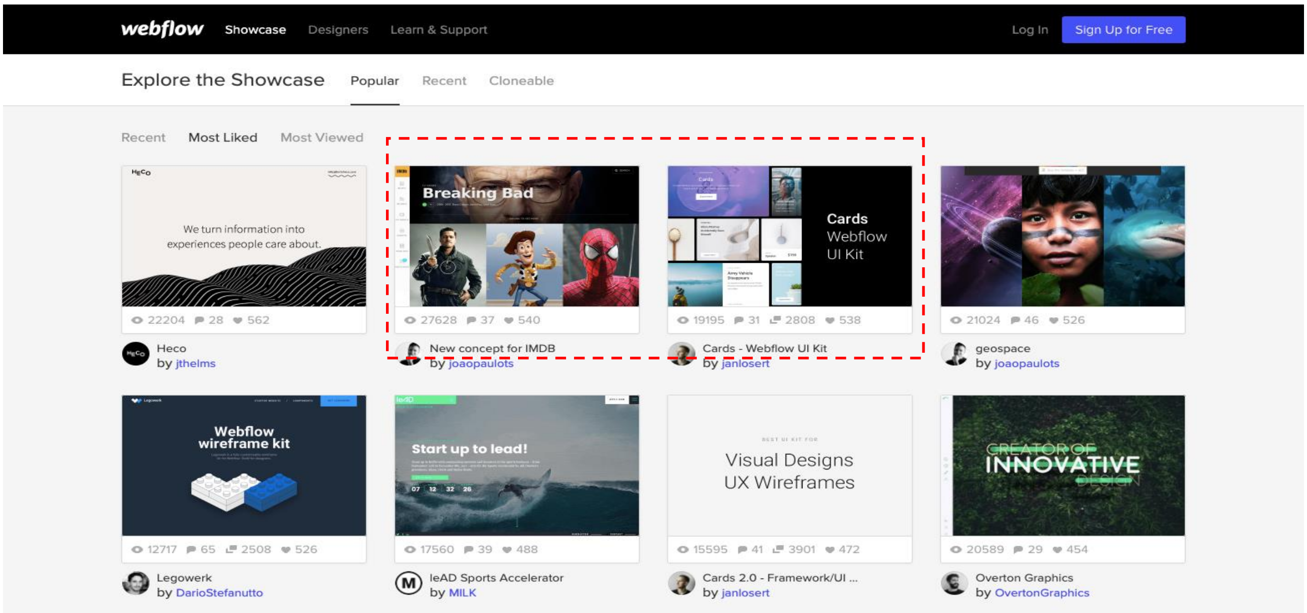
The <img> tag has two required attributes:

- src - Specifies the path to the image
- alt - Specifies an alternate text for the image, if the image for some reason cannot be displayed

```
<img src="img_girl.jpg" alt="Girl in a jacket" width="500" height="600">
```

Note: Also, always specify the width and height of an image. If width and height are not specified, the page might flicker while the image loads.

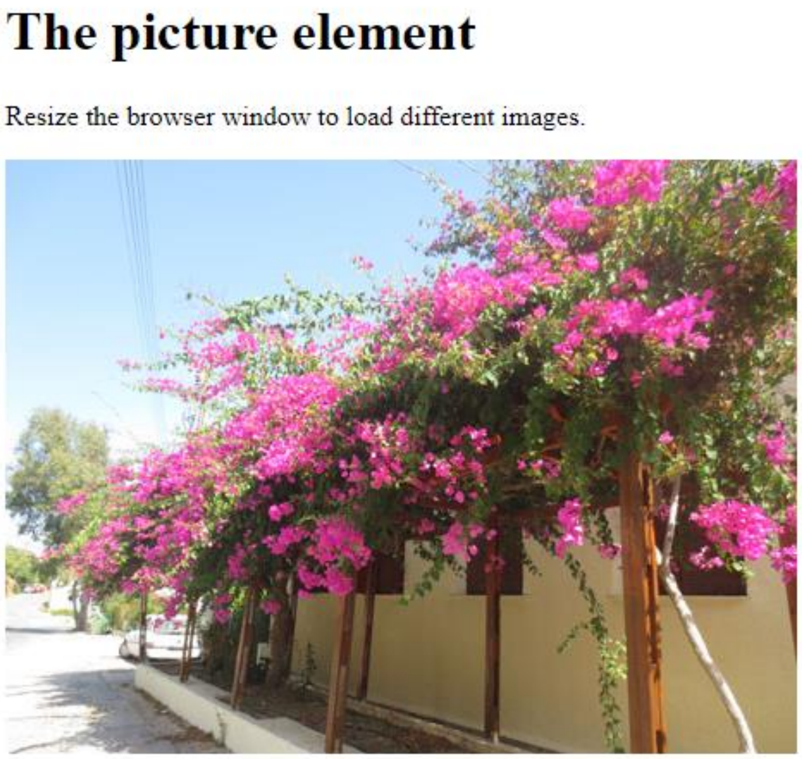
Attribute	Value	Description
alt	text	Specifies an alternate text for an image
crossorigin	anonymous use-credentials	Allow images from third-party sites that allow cross-origin access to be used with canvas
height	pixels	Specifies the height of an image
ismap	ismap	Specifies an image as a server-side image map
loading	eager lazy	Specifies whether a browser should load an image immediately or to defer loading of images until some conditions are met
longdesc	URL	Specifies a URL to a detailed description of an image
referrerpolicy	no-referrer no-referrer-when-downgrade origin origin-when-cross-origin unsafe-url	Specifies which referrer to use when fetching the image
sizes	sizes	Specifies image sizes for different page layouts
src	URL	Specifies the path to the image
srcset	URL-list	Specifies a list of image files to use in different situations
usemap	#mapname	Specifies an image as a client-side image map
width	pixels	Specifies the width of an image



HTML <picture> Tag

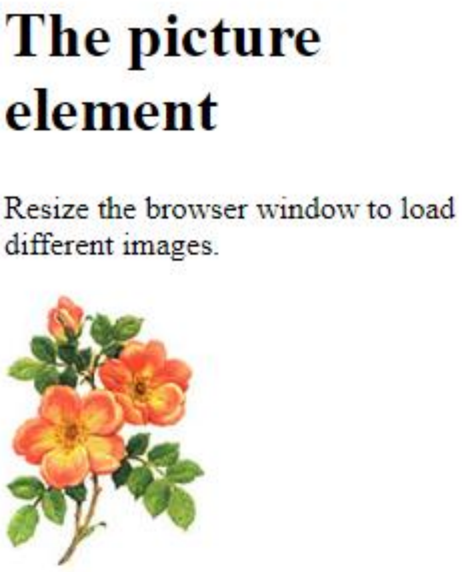
The srcset attribute specifies the URL of the image to use in different situations.
This attribute is required when <source> is used in <picture>.

```
<picture>
  <source media="(min-width:650px)" srcset="img_pink_flowers.jpg">
  <source media="(min-width:465px)" srcset="img_white_flower.jpg">
  <img src="img_orange_flowers.jpg" alt="Flowers" style="width:auto;">
</picture>
```



The picture element

Resize the browser window to load different images.



The picture element

Resize the browser window to load different images.

HTML <a> Tag

The <a> tag defines a hyperlink, which is used to link from one page to another.
The most important attribute of the <a> element is the href attribute, which indicates the link's destination.

```
<a href="https://www.iba.lk ">Visit iba.lk</a>
```

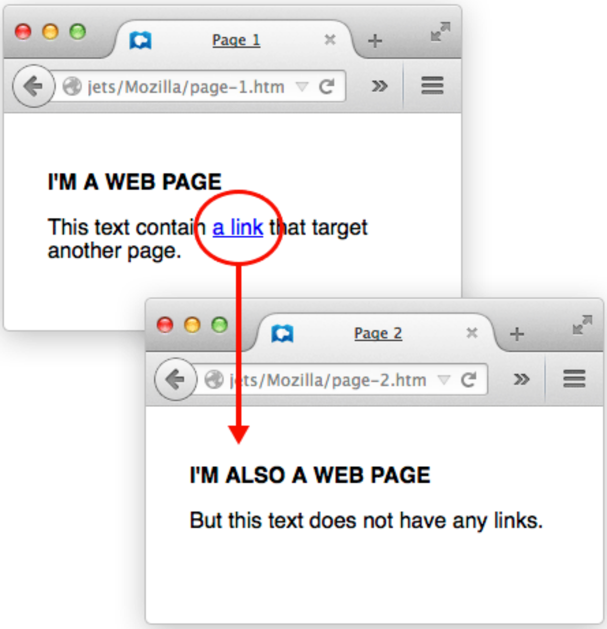
By default, links will appear as follows in all browsers:

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

Attribute	Value	Description
download	filename	Specifies that the target will be downloaded when a user clicks on the hyperlink
href	URL	Specifies the URL of the page the link goes to
hreflang	language_code	Specifies the language of the linked document
media	media_query	Specifies what media/device the linked document is optimized for
ping	list_of_URLs	Specifies a space-separated list of URLs to which, when the link is followed, post requests with the body ping will be sent by the browser (in the background). Typically used for tracking.
referrerpolicy	no-referrer no-referrer-when-downgrade origin origin-when-cross-origin same-origin strict-origin-when-cross-origin unsafe-url	Specifies which referrer information to send with the link
rel	alternate author / bookmark / external help / license / next / nofollow noreferrer / noopener / prev / search / tag	Specifies the relationship between the current document and the linked document
target	_blank _parent _self _top	Specifies where to open the linked document
type	media_type	Specifies the media type of the linked document

```
<!DOCTYPE html>  - 1.html
<html>
<body>
<h1>I'M A WEB PAGE</h1>
<p>This text contain <a href="2.html" >a
link</a> that target another page</p>
</body>
</html>
```

```
<!DOCTYPE html> - 2.html
<html>
<body>
<h1>I'M ALSO A WEB PAGE</h1>
<p>But this text does not have any links</p>
</body>
</html>
```



I'M A WEB PAGE

This text contain a link that target another page.

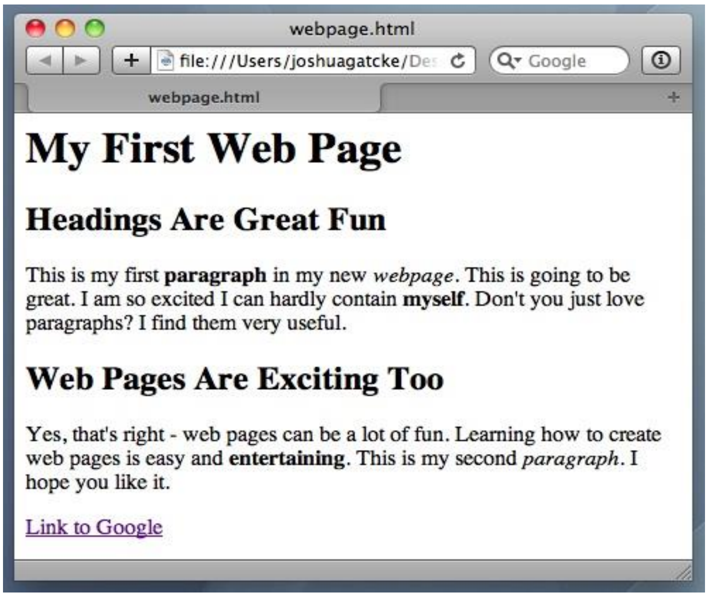
I'M ALSO A WEB PAGE

But this text does not have any links.

Exercise 01

I. Write html tags used in this web page

II. Create this web page using html tags



My First Web Page

Headings Are Great Fun

This is my first **paragraph** in my new *webpage*. This is going to be great. I am so excited I can hardly contain **myself**. Don't you just love paragraphs? I find them very useful.

Web Pages Are Exciting Too

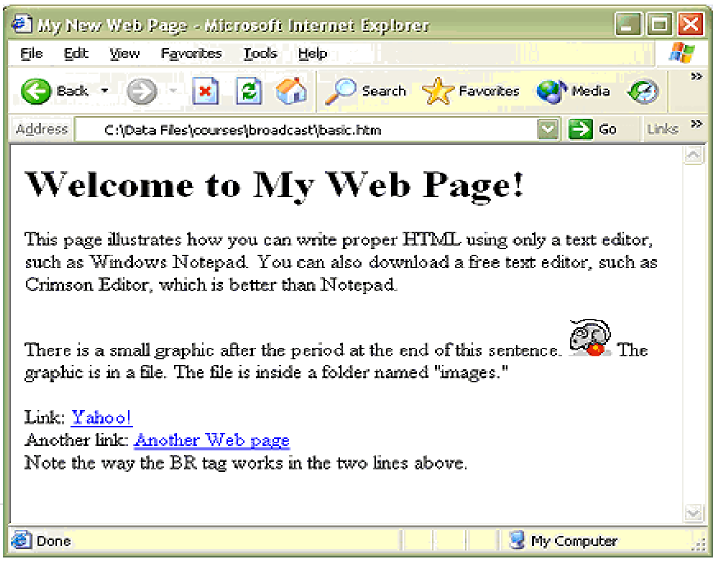
Yes, that's right - web pages can be a lot of fun. Learning how to create web pages is easy and **entertaining**. This is my second *paragraph*. I hope you like it.

Link to Google

Exercise 02

I. Write html tags used in this web page

II. Create this web page using html tags


My New Web Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Data Files\courses\broadcast\basic.htm
Go Links
Done
My Computer

Welcome to My Web Page!

This page illustrates how you can write proper HTML using only a text editor, such as Windows Notepad. You can also download a free text editor, such as Crimson Editor, which is better than Notepad.

There is a small graphic after the period at the end of this sentence. The graphic is in a file. The file is inside a folder named "images."

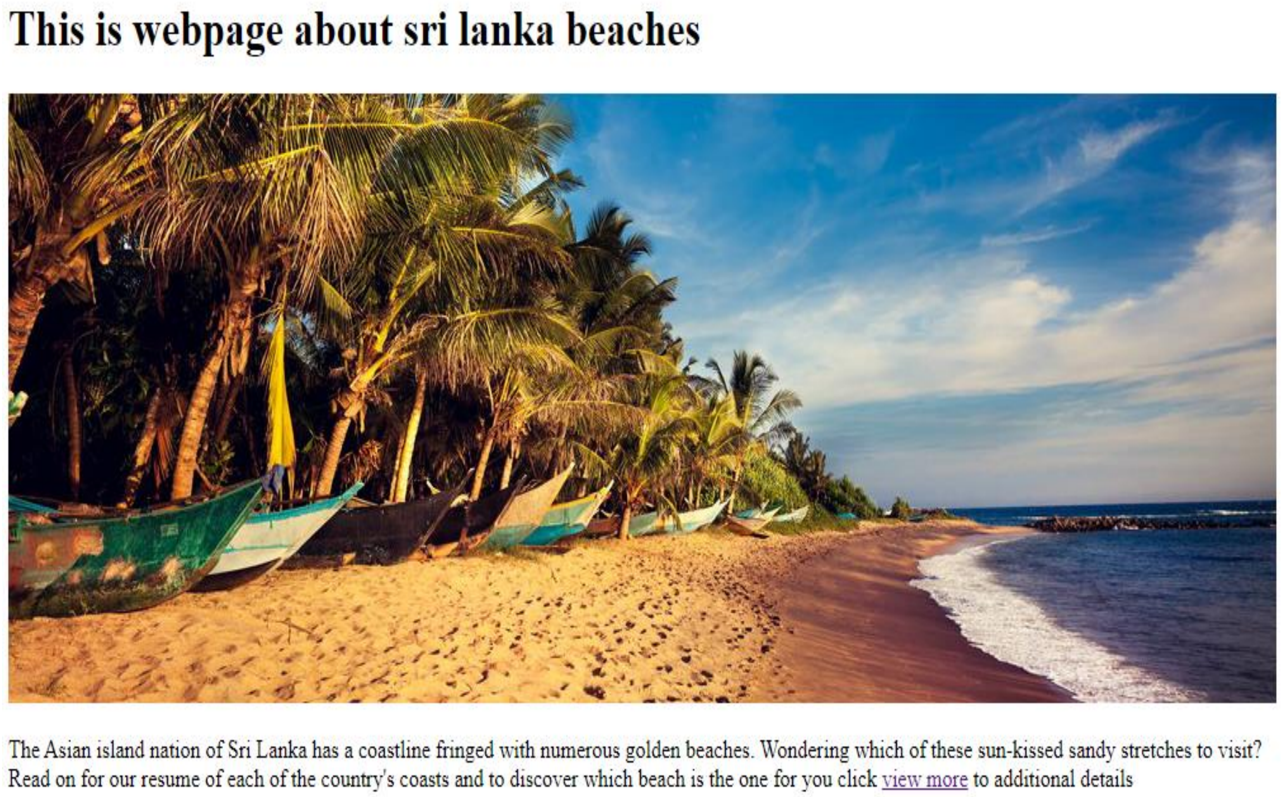
Link: Yahoo!
Another link: Another Web page
Note the way the BR tag works in the two lines above.

Exercise 03

I. Write html tags used in this web page

II. Create this web page using html tags

N.B Banner image should be responsive

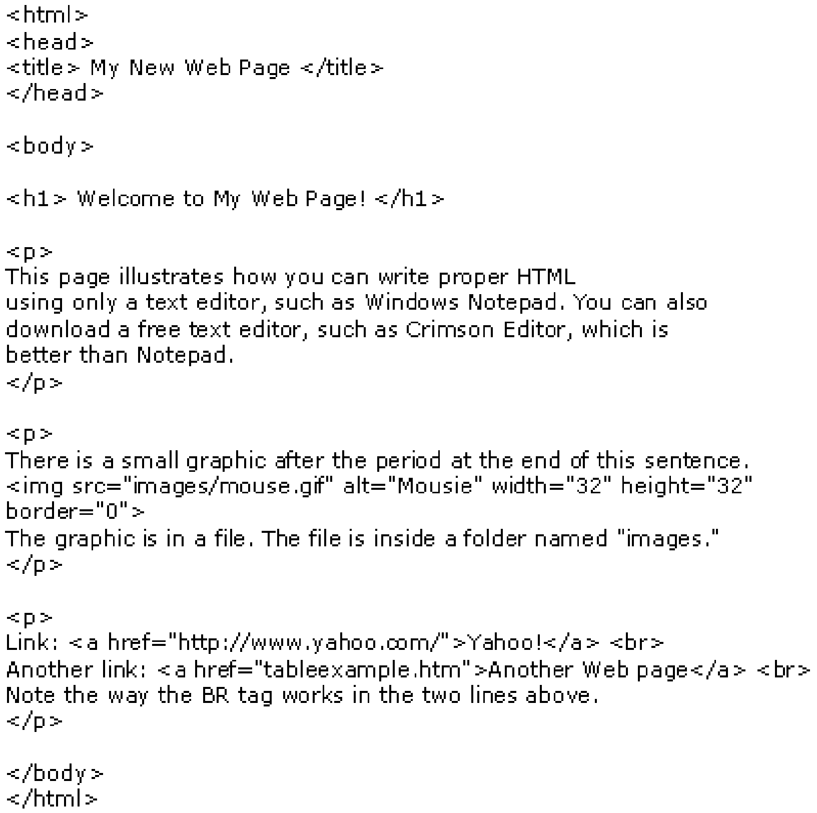


This is webpage about sri lanka beaches

The Asian island nation of Sri Lanka has a coastline fringed with numerous golden beaches. Wondering which of these sun-kissed sandy stretches to visit? Read on for our resume of each of the country's coasts and to discover which beach is the one for you click view more to additional details

Exercise 04

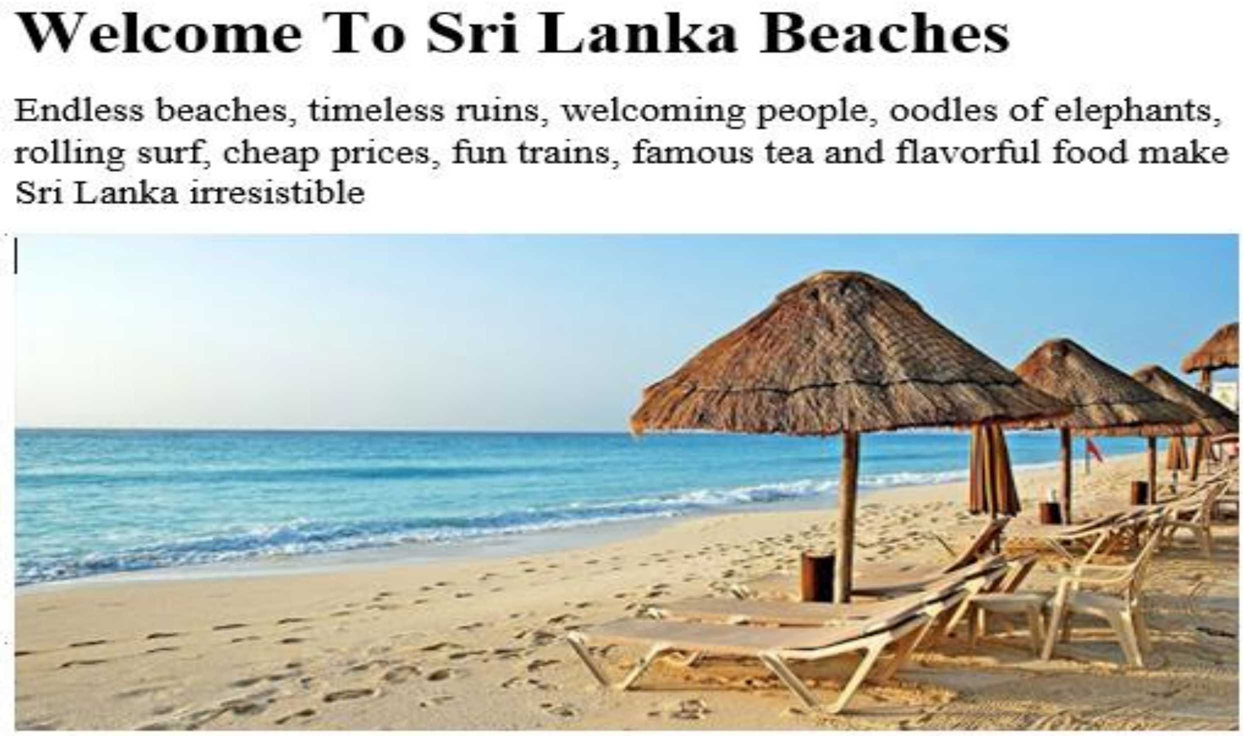
I. draw webpage display by fallowing html tags



```
<html>
<head>
<title> My New Web Page </title>
</head>

<body>

<h1> Welcome to My Web Page! </h1>

<p>
This page illustrates how you can write proper HTML
using only a text editor, such as Windows Notepad. You can also
download a free text editor, such as Crimson Editor, which is
better than Notepad.
</p>

<p>
There is a small graphic after the period at the end of this sentence.
<img src="images/mouse.gif" alt="Mousie" width="32" height="32"
border="0">
The graphic is in a file. The file is inside a folder named "images."
</p>

<p>
Link: <a href="http://www.yahoo.com/">Yahoo!</a> <br>
Another link: <a href="tableexample.htm">Another Web page</a> <br>
Note the way the BR tag works in the two lines above.
</p>

</body>
</html>
```

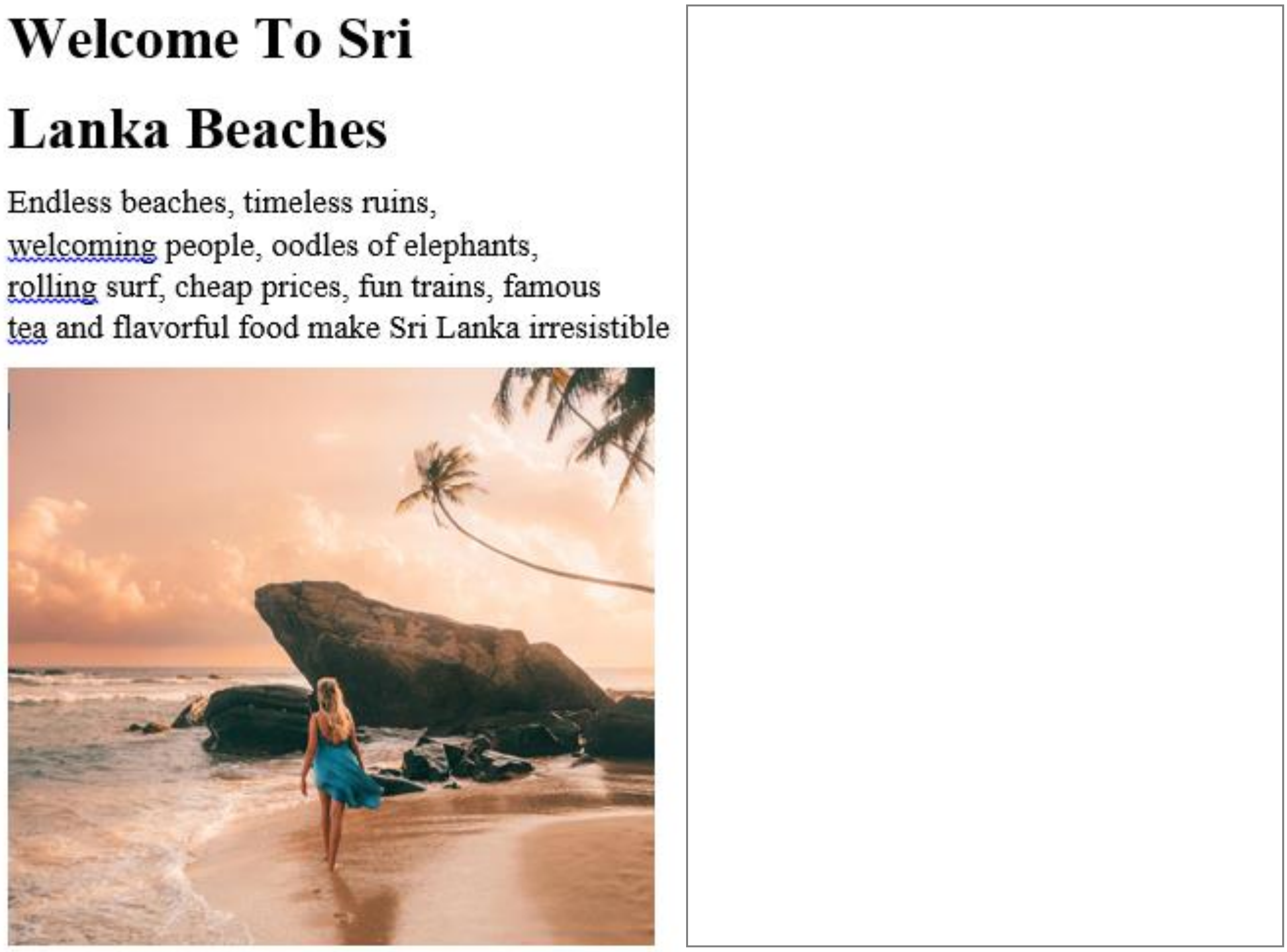
Exercise 05

I. Write html codes to design this page



Welcome To Sri Lanka Beaches

Endless beaches, timeless ruins, welcoming people, oodles of elephants, rolling surf, cheap prices, fun trains, famous tea and flavorful food make Sri Lanka irresistible



Welcome To Sri

Lanka Beaches

Endless beaches, timeless ruins,
welcoming people, oodles of elephants,
rolling surf, cheap prices, fun trains, famous
tea and flavorful food make Sri Lanka irresistible

HTML Structure and Comment Elements		
TAG (start...stop)	Attributes	Explanation
<HTML>...</HTML>		Identifies the file as containing HTML; only HEAD, BODY, and comment elements should go inside the HTML start and stop tags.
	Version="string"	Where string identifies the version of HTML; for example: <HTML Version="-//IETF//DTD HTML 3.2//EN">
<!--- string -->		Comments that browsers shouldn't display (string) can be included between these tags.

The HEAD and Related Elements		
TAG (start...stop)	Attributes	Explanation
<HEAD>...</HEAD>		brackets a set of unordered descriptive information about a document. Elements within the HEAD element include: TITLE, BASE, ISINDEX, STYLE, SCRIPT, LINK, META, and NEXTID
<TITLE> string </TITLE>		a string identifying the contents of the document; may not contain anchors, paragraph elements, or highlighting; every HTML must have one TITLE element.
<BASE>...</BASE>		used to record the URL of the original version of a document; useful for when the source file is transported elsewhere
	Href="URL"	defines base URL of the document;
<ISINDEX>		marks the document as searchable--the server on which the document is located must have a search engine defined that supports this searching.
<STYLE>		a way for the author of a document to define rendering information which will work with style sheets.
<SCRIPT>		reserved for future use with scripting languages.
<LINK>		used to define a relationship between the document and other objects or documents.
	Href="URL	this identifies the document or part of a document to which this link refers.
	Name="rel|rev"	this is a way to name this LINK as a possible destination for another hypertext document;
	Rel="made|..."	describes the relationship defined by this LINK, according to the possible relationships as defined by https://www.w3.org/hypertext/WWW/MarkUp/Relationships.html. For example, the value "made" indicates authorship.
	Rev="made|..."	similar to rel, above, but the rev attribute indicates the reverse relationship as Rel. For Example, the LINK with Rel="made" indicates that the Href

		attribute indicates the URL given in the Href is the author of the current document. Using the Rev="made" link indicates that the current document is the author of the URL given in the Href attribute.
	Urn="string"	indicates the Uniform Resource Name of the document; the specification for URN and other addressing is still in development (https://www.w3.org/hypertext/WWW/Addressing/Addressing.html).
	Title="string"	this attribute is not to be used as a substitute for the TITLE attribute of the document itself, but as a title for the document given by the Href attribute of the LINK element. This attribute is rarely used or supported by browsers, but may have value for cross referencing the relationships the LINK element defines.
	Methods="..."	describes the HTTP methods the object referred to by the Href of the LINK element supports. For example, one method is searching; a browser could thus use this Methods attribute to give information to the user about the document defined by the LINK element.
<META>		used to identify meta-information (information about information) in the document. This element is not meant to take the place of elements which already have a specific purpose.
	Http-equiv="..."	This attribute connects this META element to a particular protocol response which is generated by the HTTP server hosting the document.

	Http-equiv="refresh"
	Content="n"
		browser will reload the currently displayed page after n seconds.

	Http-equiv="refresh"
	Content="n; Url=URL"
		browser will load the document at URL after displaying the currently displayed page
		for n seconds.

	Http-equiv="expires"
	Content="string-date"
		an expiry date for files that are periodically updated. string-date could be, for example, "17-Jan-1999 15:00:00 GMT"

	Name="string"	This attribute is a name for the information in the document--not the title of the document (which should be defined in the TITLE element) but a "meta name" classifying this information.
	Content="string"	A "meta name" for the content associated with the given name (defined by the Name attribute) or the response defined in Http-equiv.
<NEXTID>		This element is used by text generated software in creating identifiers;
	N="string"	used to define the next identifier to be allocated by the text generator program. Normally, human writers of HTML don't use this element; and Web browsers ignore this element.

The BODY and Related Elements

TAG (start...stop)	Attributes	Explanation

<BODY></BODY>		delimit the content of an HTML document
	Background = "URL"	the URL of the graphic that will be tiled as the background of the page
	BGColor="color"	the color of the background of the page;
	Text="color"	the color of the document's text.
	Link="color"	the color of the document's hotspots
	VLink="color"	the color of the document's visited links
	ALink="color"	the color of the document's hotspots during user selection
<A>...</A>		the anchor element which is used as the basis for linking documents together.
	Href="URL"	this attribute identifies the URL of the hypertext reference for this anchor in the form Href="URL", where the URL given will be the resource which the browser retrieves when the user clicks on the anchor's hotspot.
	Name="string"	this attribute creates a name for an anchor; this name can then be used within the document or outside of the document in anchor to refer to the portion of text identified by the name.
	Title="string"	this attribute is for the title of the document given by the Href attribute of the anchor. A

		browser could use this information to display this title before retrieving it, or to provide a title for the Href document when it is retrieved (e.g., if the document is at a FTP site, it will not have a title defined).
	Rel="made|..."	Defines the relationship defined from the current document to the target (Href document). See the discussion of the Rel attribute in the LINK element, above.
	Rev="made|..."	Defines the relationship defined from the target (Href document) to the current document. See the discussion of the Rev attribute in the LINK element, above.
	Urn="string"	This indicates the Uniform Resource Name of the target (Href) document; the specification for URN and other addressing is still in development.
	Methods="..."	Provides information about the functions the user can perform on the Href object. Similar to described above for the Title attribute, this information might be useful for the browser to display in advance.
Character blocks: elements that "chunk" text in lists or blocks		
<DIV>...</DIV>		used to define a block or paragraph of text;

	Align="left|right"	aligns the block or paragraph of text
<BLOCKQUOTE>...</BLOCKQUOTE>		
<PRE>...</PRE>		sets up a block of text which will be presented in a fixed-width font, with spaces as significant.
<BLOCKQUOTE>...</BLOCKQUOTE>		brackets text that is an extended quotation from another source.
<UL>...</UL> <OL>...</OL> <MENU>...</MENU> <DIR>...</DIR>		Lists for information; all use the LI element to identifiy the elements. UL brackets an unordered list of items; OL brackets an ordered list of items; MENU brackets an unordered list of items; DIR brackets a list of items which are at most 20 characters wide;
	Compact	makes the list compact
	Type="disc|circle|square"	defines the bullet type for a UL list;
	Type="1|a|A|i|I"	defines the bullet type for a OL list; 1 (arabic numbers: 1, 2, 3, ...); a (lower alpha: a, b, c, ...) A (upper alpha: A, B, C, ...) i (lower roman: i, ii, iii, ...) I (upper roman: I, II, III, ...)
	Start="N"	starts the OL list off with a sequence starting at N.
<LI>		identifies a list element in UL, OL, MENU, DIR.
<DL>...</DL>		A definition list, or glossary; uses DT to identify terms and DD to identify definitions.

	Compact	makes the list compact.
<DT>		identifies term in definition list (DL).
<DD>		identies description in definition list (DL).
<ADDRESS>...</ADDRESS>		ownership or authorship information, typically at the start or end of a document.
Headers		
<H1>...</H1>		Level 1 Header
<H2>...</H2>		Level 2 Header
<H3>...</H3>		Level 3 Header
<H4>...</H4>		Level 4 Header
<H5>...</H5>		Level 5 Header
<H6>...</H6>		Level 6 Header
Separators		
<HR>...</HR>		creates a horizontal rule.
	Size="n"	this attribute identifies the thickness of the line.
	Noshade	this attribute turns off shading to create a solid bar.
	Width="n| n%"	this attribute identifies the width of the line, either expressed as width in pixels, or a relative with as a percent of the current display width (not page width). These lines are by default centered (default can be overridden with the Align attribute).

	Align="left| right| center"	specifies the alignment of horizontal lines that are less than the full width of the page.
<P>		identifies start of paragraph; the stop tag </P> is optional.
	Align="center|right|left"	aligns a paragraph;
Spacing		
<BR>		forces a linebreak. Typically, this is used to represent postal addresses or text; where linebreaks are significant.
Alignment		
<CENTER>string</CENTER>		centers text.
Images		
<IMG>...</IMG>		places graphic image in a document at the location of the element tag (an "inline image").
	Src="URL"	identifies the source file of the image.
	Alt="string"	a string of characters can be defined that will be displayed in non-graphical browsers. Non-graphical browsers otherwise ignore the IMG element.
	Align="top|middle|bottom|left|right"	sets the positioning relationship between the graphic and the text that follows it; values include: top: the text following the graphic should be aligned with the top of the graphic. middle: the text following the

		graphic should be aligned with the middle of the graphic. bottom: the text following the graphic should be aligned with the bottom of the graphic. left: the image is pushed to the left margin of the page. right: the image is pushed to the right margin of the page.
	Width="n"	defines the width of the image.
	Height="n"	defines the height of the image.
	Border="n"	defines the border above and below the image.
	Vspace="n"	defines the vertical space around the image.
	Hspace="n"	defines the horizontal space to the left and right of the image.
	Ismap	this attribute identifies the image as an image map, where regions of the graphic are mapped to defined URLs. Hooking up these relationships requires knowledge of setting an image map file on the server to define these connections.
	Usemap="string"	identifies which MAP element, as defined in the Name attribute, that you are using for a client-side image map.
<MAP>...</MAP>		define client-side image maps
	Name="string"	string is the name you use as the value of the Usemap attribute of

		the corresponding IMG element.
<AREA>...</AREA>		defines correspondence between pixels and URLs in client side image map element MAP used in conjunction with IMG
	Shape="rect|poly|circle"	Sets the shape of the hotspot in the image
	Shape="rect" Coords="left,top,right,bottom"	Sets the perimeter x-y coordinates of a rectangle
	Shape="poly" Coords="x1,y1,x2,y2,...xn,yn""	Sets the border coordinates of a polygon
	Shape="circle" Coords="x,y,r"	Sets the center (x, y) and radius (r) of a circle
	Href="URL"	Defines the URL associated with this AREA
	Nohref	Defines this AREA as a "deadspot," with not URL associated with it
<FIG>...</FIG>		define a figure
	Src="URL"	defines the source of the figure
Character Formatting		
TAG (start...stop)	Attributes	Explanation
<CITE>string</CITE>		delimits a citation.
<CODE>string</CODE>		delimits computer language source code.
<EM>string</EM>		delimits emphasized text.
<KBD>string</KBD>		delimits text that is intended to be entered as a keyboard entry.
<SAMP>string</SAMP>		delimits text that should be rendered "as is."
<STRONG>string</STRONG>		delimits text with a strong emphasis.

<VAR>string</VAR>		delimits a variable name.
<DFN>string</DFN>		delimits the defining instance of an item.
<Q>string</Q>		delimits a short quote.
<LANG>string</LANG>		the human language currently defined.
<AU>string</AU>		the name of an author.
<PERSON>string</PERSON>		the name of a human.
<ACRONYM>string</ACRONYM>		an acronym in the document.
<INS>string</INS>		inserted text (when documents are amended).
<DEL>string</DEL>		deleted text (when documents are amended).
<B>string</B>		delimits bold text.
<I>string</I>		delimits italics text.
<TT>string</TT>		delimits typewriter font text.
<U>string</U>		delimits underlined text.
<BIG>string</BIG>		big print relative to current font.
<SMALL>string</SMALL>		small print relative to current font.
<SUB>string</SUB>		a subscript.
<SUP>string</SUP>		a superscript.
<BASEFONT>...</BASEFONT>		changes the current font.
	Size="n"	specifies the font size in the range 1-7.
	Size="+|-n"	specifies font size relative to the current base font size
	Color="color"	specifies the font color; either RGB hexadecimal value or color name (see BODY attribute BGColor).

<FONT> string </BASEFONT>		changes the font for string.
	Size="n"	specifies the font size in the range 1-7.
	Size="+|-n"	specifies font size relative to the current base font size
	Color="color"	specifies the font color; either RGB hexadecimal value or color name (see BODY attribute BGColor).
FORM Elements		
TAG (start...stop)	Attributes	Explanation
<FORM>...</FORM>		delimits the content of a FORM.
	Action="URL"	this attribute identifies the URL of the program or script which accepts the contents of the form for processing. If this attribute is absent, the BASE URI of the Form is used.
	Method="get|post"	this attribute this attribute indicates the variation in the Froms-handling protocol which will be used in processing the Action program or script.
	Encytpe="string"	this attribute identifies the media type (See RFC1590) that will be used for encoding the name/value pairs of the Form's data. This is needed for when the protocol identified in Method does not have its own format. The default encoding for all Forms is

		application/x-www-form-urlencoded.
<INPUT>...</INPUT>		used for collecting information from the user.
	Align="top|middle|bottom"	this attribute is used used only with the image Type (see list below). Possible values are "top," "middle," and "bottom," and define the relationship of the image to the text following it.
	Checked	this attribute causes the initial state of a checkbox or radio button to be "selected." Without this attribute, the initial state is unselected.
	Maxlength="n"	his attribute sets a maximum number of characters that a user can enter in a text a field. The default value of this is unlimited.
	Name="string	this attribute identifies the symbolic name that is used in transferring and identifying the output from this element of the Form.
	Size="n"	this attribute specifies the field width as displayed to the user. If Size is less than Maxlength, the text field is scrollable.
	Src="URL"	this attribute the source file for the image used with the attribute Type is set to "image."
	Type="checkbox| hidden| image| password| radio| reset| submit| text"	this attribute identifies the type of the input field: checkbox: This is used for gathering

		data that can have multiple values at a time. hidden: This is for values that are set by the form without input from the user. image: An image field be used to submit the Form: when the user clicks on the image, the Form is submitted, and the x and y coordinates of the click location are transmitted with the name/value pairs. password: This is a field in which the user enters text, but the text is not displayed (could appear as stars). radio: Used to collect information where there is one and only one possible value from a set of alternatives. The Checked attribute can set the initial value of this element. reset: This is used to reset and clear the Form to its default values. The Value attribute sets the string displayed to the user for this element. submit: This button is used to submit the Form. The Value attribute sets the string displayed to the user for this element. text: This is used for a single line of text; this uses the Size and Maxlength attributes. For multiple lines, use TEXTAREA (below).

	Text	this attribute identifies the input as a single line text-entry area.
	Value="string"	this attribute sets the initial displayed value of the field or the value of the field when it is selected (the radio button type must have this attribute set).
<SELECT>...</SELECT>		used for presenting a user with a choice of a set of alternatives. The OPTION element is used to define each alternative.
	Name="string"	identifies the logical name that will be submitted and associated with the data as a result of the user choosing select.
	Multiple	By default, the user can only make one selection from the group in the SELECT element. By using the Multiple attribute, the user may select one or more of the OPTIONs.
	Size="n"	specifies the number of visible items. If this is more than one, the visual display will be a list.
<OPTION>...</OPTION>		occurs only within the SELECT element (above) and is used to represent each choice of the SELECT.
	Selected	indicates that this option is initially selected.
	Value="n"	If present, this is the value that will be returned by the SELECT if this option is chosen; otherwise, the value

		returned is that set by the OPTION element.
<TEXTAREA>...</TEXTAREA>		used to collect multiple lines of text from the user; the user is presented with a scrollable pane in which text can be written.
	Name="string"	identifies the logical name that will be associated with the returned text.
	Rows="n"	the number of rows of text that will be displayed (the user can use more rows and scroll down to them).
	Cols="n"	the number of columns of text that will be displayed (the user can use more columns and scroll to the right to them).
TABLE and Related Elements		
TAG (start...stop)	Attributes	Explanation
<TABLE>...</TABLE>		delimits the content of a TABLE.
	Align="bleedleft| left| center| right| bleedright| justify "	the horizontal alignment of the table on the screen (not the contents of the table). Possible values are: bleedleft: aligned at the left window border left: at the left text margin center: centered between text margins right: at the right text margin bleedright: aligned at the right window border justify: table should fill space between text margins

	Border	this attribute causes browser to render a border around the table; if missing, the table has no grid around it or its data.
	Border="n"	specifies the thickness of the table border in pixels.
	Cellspacing="n"	specifies the space around data cells in pixels.
	Cellpadding="n"	specifies the space data in the cells in pixels.
	Width="n"	this attribute specifies how wide the table will be; if given as "N%", the width is N% of the width of the display.
	Id="string"	a document-wide identifier for naming positions in the document as distinations for a hypertext link.
	Class = "string"	A list of names that may be used by style sheets.
	Lang = "string"	the natural (human) language used by the content of the table.
	Dir="ltr|rtl|"	identifies the layout of rows--such as column 1 is on the right (Dir="rtl") or left (Dir="ltr"). Used for languages that have different directionality in reading (not left to right, top to bottom).
	Align="left|center|right|justify|char"	the horizontal alignment of cell contents; justify is left; char is for aligning on a character
	Char="c"	specifies the alignment character for use with Align="char";

	Charoff="n"	the offset of the alignment character on each line.
	Valign="top|middle|bottom|baseline"	alignment of cell contents
	Cols="n"	the number of columns in the table; if present, the browser can render the table as the data is received;
	Frame="(void| above| below| hsides| lhs| rhs| vsides| box| border"	which sides of the frame to render (void = none).
	Rules="(none | groups | rows | cols | all")	where to draw the rules in the table interior.
<TR>...</TR>		contains the elements in each table row.
<TH>string</TH> <TH>string</TH>		TH used to identify a heading in the table; TD used to identify data in the table.
	Align="left| center| right| justify| decimal"	this attribute identifies horizontal alignment of the items in a table row.
	Valign="top | middle | bottom| baseline"	this attribute identifies the vertical alignment of the items in a table cell.
	Colspan="n"	this attribute identifies the number of columns the cell spans.
	Rowspan="n"	this attribute identifies the number of rows the cell spans.
	Nowrap	this attribute prevents the browser from wrapping the contents of the cell.
<CAPTION>string</CAPTION>		used to label a table or figure.
	Align="top| bottom| left| right"	this attribute identifies the position of the caption relative to the table or figure.
Netscape Extension Elements		

TAG (start...stop)	Attributes	Explanation
<ISINDEX>...</ISINDEX>	Prompt="string"	the message a user sees for a searchable index.
<BLINK>string</BLINK>		creates blinking text.
<LI> in <UL>	Type="disc|circle|square"	changes shape of individual items in UL element
<LI> in <OL>	Type="A|a|I|i|1"	changes shape of individual items in OL element; A = capital letters; a = small letters; I = capital roman numerals; 1 = numbers (default)
<LI Value="n"> in <OL>	Value="n"	changes value of item in OL element;
<IMG>...</IMG>		additions to the IMG element.
	Align="left| right| top| texttop| middle| absmiddle| baseline| bottom| absbottom"	specifies the placement of an image relative to the text following it.
Frames Elements (Netscape and Microsoft support)		
TAG (start...stop)	Attributes	Explanation
<FRAMESET> ... </FRAMESET>		container for frame elements; used instead of a body in a document
	Rows="string,string,..string"	comma-separated list of values indicating the height of each of the rows. If a number, the row is that high in pixels; If a number followed by a %, the row gets that percent of the total window height; If a *, the row gets the remaining room; If a number followed by a *, that row gets that much more share of the height;
	Cols="string,string,..string"	comma-separated list of values indicating the height of each of the

		columns. The values are the same syntax as for the Rows attribute.
<FRAME> ... </FRAME>		this element identifies a single frame in a frameset
	Src="URL"	The url refers to the resource to be initially displayed in this frame.
	Name="string"	assigns a name to the frame; this name then can be used in other documents in the Target attribute of the anchor element.
	Marginwidth="n"	The number of pixels to add to the left and right of the contents of frame the frame.
	Marginheight="value"	The number of pixels to add to the top and bottom of the contents of frame the frame.
	Scrolling="yes|no|auto"	yes = add scrollbars, even if they are not needed; no = NEVER add scrollbars even if they are needed; auto = add scrollbars if they are needed; (default)
	Noresize	a flag that indicates the frame is not resizable by the user; Normally, a user can manually alter the size of the frame using "grab buttons" that appear on the display of the frame. The Noresize attribute makes this resizing impossible.
<NOFRAMES> ... </NOFRAMES>		this element brackets content that will be rendered by non-frame-enabled browsers.

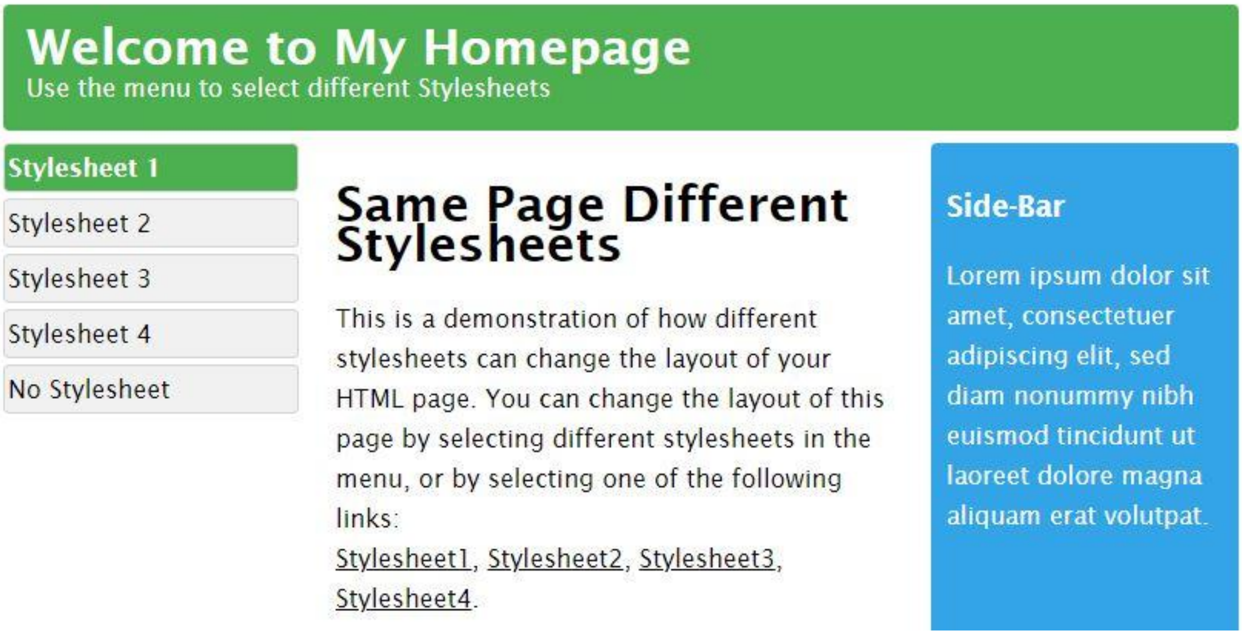
<A>	Target="string"	lets you define in which frame the new content referenced in the anchor will be displayed when selected. Target has the possible values: name: a frame named in a FRAME element's Name attribute; _self: new document is displayed the in same frame as the anchor that loads it; this is the default; _parent: displays the new document in the parent frame; if no parent, same as _self; _top: displays the new document in the entire window; if no frames, same as _self; _blank: display the new document in a new, unnamed window;
<BASE>	Target="string"	lets you set the default target for every hypertext link in the document. Its possible values are the same as for the Target attribute of the A element.
Microsoft Internet Explorer Extension Elements		
TAG (start...stop)	Attributes	Explanation
<MARQUEE> ... </MARQUEE>		creates a scrolling text marquee
	Align="top|bottom|middle"	alignment of marquee
	Behavior="scroll|slide|alternate"	(scroll = continous movement; slide = display once and then sit there; alternate = slide in and reverse out)
	Direction="left|right"	scroll direction

	Loop="n"	how many times to scroll
	Scrollamount="n"	scrollrate in pixels/time
	Scrolldelay="n"	milleseconds between scroll movements
	BGColor="color"	color of background behind text
	Width="n"	width in pixels of marquee
	Height="n"	height in pixels of marquee
	Hspace="n"	pixels to leave as buffer left and right of marquee
	Vspace="n"	pixels to leave as buffer above and below marquee
<BGSOUND> ... </BGSOUND>		plays a soundtrack when the document is displayed
	Src="URL"	url of sound file
	Loop="n"	number of times to replay file"
<IMG>	new attributes for IMG element	
	Dynsrc="URL"	URL of the AVI movie
	Controls	when present, it adds "VCR-like" controls to the movie image.
	Loop="n"	number of times to play the movie
	Start="mouseover|fileopen"	when movie begins to play, fileopen is default.
<BODY>		new attributes for the BODY element
	BGProperties="fixed"	if present, this freezes the background image defined in Background to the browser window so that it does not scroll with the text.
	Topmargin="n"	pixels of buffer at top of page in browser window

	Leftmargin="n"	pixels of buffer at the left of page in browser window
<TABLE>	BGColor="color"	specifies the background color of the entire table
<TR>	BGColor="color"	specifies the background color of the table row
<TH>	BGColor="color"	specifies the background color of the table header
<TD>	BGColor="color"	specifies the background color of the table data

Introduction to CSS

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files



Welcome to My Homepage

Use the menu to select different Stylesheets

Same Page Different Stylesheets

This is a demonstration of how different stylesheets can change the layout of your HTML page. You can change the layout of this page by selecting different stylesheets in the menu, or by selecting one of the following links: Stylesheet1, Stylesheet2, Stylesheet3, Stylesheet4.

Side-Bar

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Why Use CSS?

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

```
body {
  background-color: lightblue;
}
h1 {
  color: white;
  text-align: center;
}
p {
  font-family: verdana;
  font-size: 20px;
}
```

HTML was NEVER intended to contain tags for formatting a web page!

HTML was created to describe the content of a web page, like:

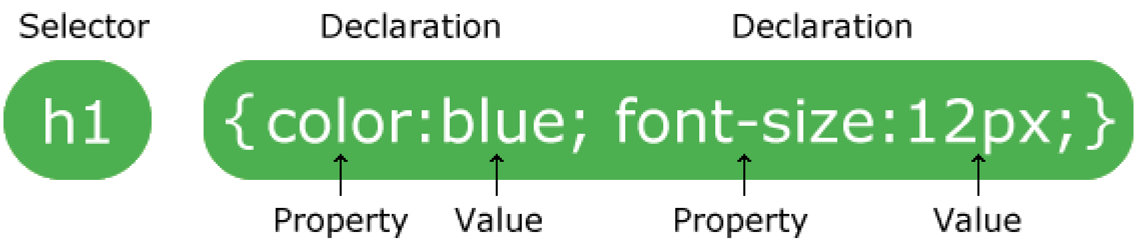
<h1>This is a heading</h1>

<p>This is a paragraph.</p>

When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large websites, where fonts and color information were added to every single page, became a long and expensive process.

CSS Syntax

A CSS rule-set consists of a selector and a declaration block:



The selector points to the HTML element you want to style.
The declaration block contains one or more declarations separated by semicolons.
Each declaration includes a CSS property name and a value, separated by a colon.
Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

```
p {
  color: red;
  text-align: center;
}
```

- **p** : is a selector in CSS (it points to the HTML element you want to style: <p>).
- **color** : is a property, and red is the property value
- **text-align** : is a property, and center is the property value

CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

- **Simple selectors** (select elements based on name, id, class)
- **Combinator selectors** (select elements based on a specific relationship between them)
- **Pseudo-class selectors** (select elements based on a certain state)
- **Pseudo-elements selectors** (select and style a part of an element)
- **Attribute selectors** (select elements based on an attribute or attribute value)

The CSS element Selector

The element selector selects HTML elements based on the element name.

Here, all <p> elements on the page will be center-aligned, with a red text color:

```
p {
  text-align: center;
  color: red;
}
```

The CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element! To select an element with a specific id, write a hash (#) character, followed by the id of the element.

The CSS rule below will be applied to the HTML element with id="para1":

```
#para1 {
  text-align: center;
  color: red;
}
```

Note: An id name cannot start with a number!

The CSS class Selector

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name. In this example all HTML elements with class="center" will be red and center-aligned:

```
.center {
  text-align: center;
  color: red;
}
```

You can also specify that only specific HTML elements should be affected by a class.

In this example only <p> elements with class="center" will be center-aligned:

```
p.center {
  text-align: center;
  color: red;
}
```

The CSS Universal Selector

The universal selector (*) selects all HTML elements on the page.

The CSS rule below will affect every HTML element on the page:

```
* {
  text-align: center;
  color: blue;
}
```

The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

```
h1 {
  text-align: center;
  color: red;
}
```

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External CSS
- Internal CSS
- Inline CSS

External CSS

With an external style sheet, you can change the look of an entire website by changing just one file!

Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

External styles are defined within the <link> element, inside the <head> section of an HTML page:

```
<!DOCTYPE html>
<html>
        <head>
        <link rel="stylesheet" href="mystyle.css">
        </head>
        <body>
        <h1>This is a heading</h1>
        <p>This is a paragraph.</p>
</body>
</html>
```

An external style sheet can be written in any text editor, and must be saved with a .css extension.

The external .css file should not contain any HTML tags.

Here is how the "mystyle.css" file looks:

Note: Do not add a space between the property value and the unit (such as margin-left: 20 px;). The correct way is: margin-left: 20px;

Internal CSS

An internal style sheet may be used if one single HTML page has a unique style.

The internal style is defined inside the <style> element, inside the head section.

Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: linen;
}

h1 {
  color: maroon;
  margin-left: 40px;
}
</style>
</head>
```

Inline CSS

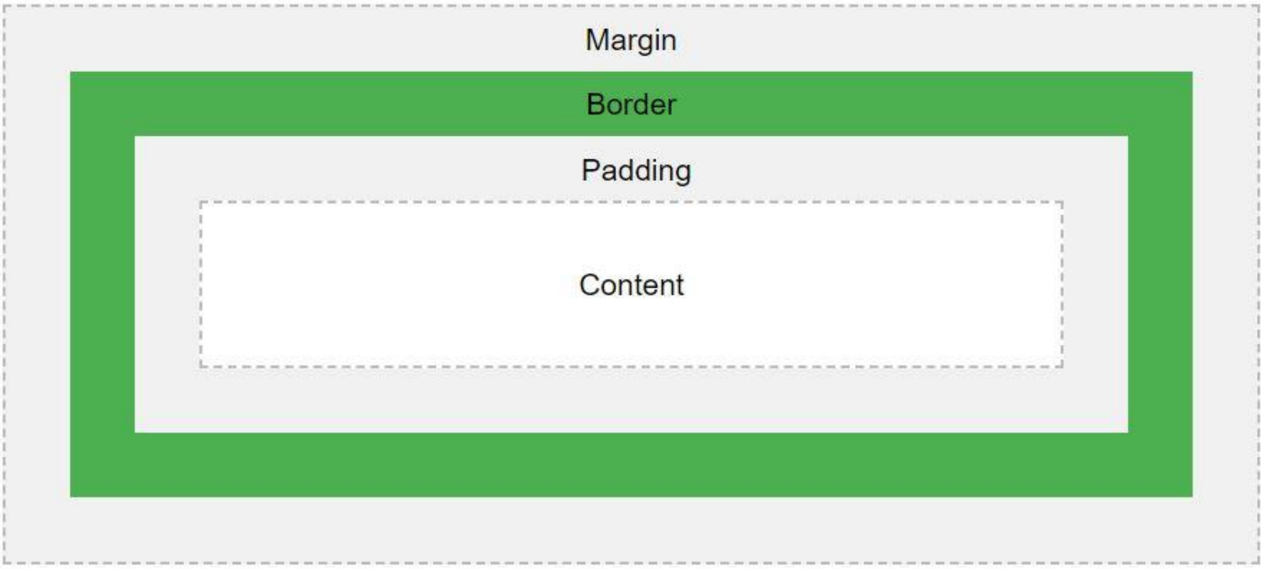
An inline style may be used to apply a unique style for a single element.
To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.
Example
Inline styles are defined within the "style" attribute of the relevant element:

```
<h1 style="color:blue;text-align:center;">This is a heading </h1>
<p style="color:red;">This is a paragraph.</p>
```

The CSS Box Model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model



Explanation of the different parts:

- **Content -** The content of the box, where text and images appear
- **Padding -** Clears an area around the content. The padding is transparent
- **Border -** A border that goes around the padding and content
- **Margin -** Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

Demonstration of the box model:

```
div {
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
```

CSS Selectors Cheat Sheet for Developers

Selector	Syntax	Description	Example
Universal Selector	*	Selects all the elements	* {color: pink;font-size: 20px;}
Type Selector	element{properties}	Selects elements based on element type	p {color: pink;font-size: 20px;}
Id Selector	#id{properties}	Selects an element with a specified id	#adbox {width: 80px;margin: 5px;}
Class Selector	.class{properties}	Selects elements of the specified CSS class	.dark {color: black;}
Attribute Selectors			
	element[attribute]{properties}	Selects elements with the specified attribute	input[disabled] {background-color: #fff;}
	element[attribute="value"]	Selects elements with the specified attribute equaling a value	input[type=”text”] {color: black;}
	element[attribute~="value"]	Selects elements with the specified attribute the value of which contains a specific word.	h1[title~="Codota"] {color: pink;font-size: 20px;}
	element[attribute|="value"]	Selects elements with the specified attribute and attribute value with the attribute beginning with specified value (or specified value immediately followed by “-“)	a[hreflang|="en"] { color: blue;}
	element[attribute^="value"]	Selects elements with the specified attribute with the attribute value starting with specified value	h2[title^="Codota"] {color: black;font-size: 20px;}
	element[attribute$="value"]	Selects elements with the specified attribute with the attribute value ending with specified value	h3[title$="Right Now!"] {color: red;font-size: 30px;}
	element[attribute*="value"]	Selects elements with the specified attribute where the attribute value contains a specified value	h4[title*="new"] {color: red;font-size: 20px;}

Descendant Combinator	element1 element2{properties}	Selects all specified child descendant elements (element2) under the parent element (element1).	div p{color:pink;}
Child Combinator	element1 > element2{properties}	Selects all specified immediate child elements (element2) under the parent element (element1).	div > p{color:pink;}
Pseudo Classes			
	element:link{properties}	Selects unvisited link elements	a:link{color:blue;}
	element:visited{properties}	Selects visited link elements	a:visited{color:red;}
	element:active{properties}	Selects active link elements	a:active{color:green;}
	element:hover{properties}	Selects mouseover hover elements	a:hover{color:purple;}
	element:focus{properties}	Selects in-focus elements	a:focus{color:pink;}

CSS Properties Order by Category

The following section contains a complete list of standard properties belonging to the latest CSS3 specifications. All the properties are listed alphabetically.

Property	**Description**
align-content	Specifies the alignment of flexible container's items within the flex container.
align-items	Specifies the default alignment for items within the flex container.
align-self	Specifies the alignment for selected items within the flex container.
animation	Specifies the keyframe-based animations.
animation-delay	Specifies when the animation will start.
animation-direction	Specifies whether the animation should play in reverse on alternate cycles or not.
animation-duration	Specifies the number of seconds or milliseconds an animation should take to complete one cycle.
animation-fill-mode	Specifies how a CSS animation should apply styles to its target before and after it is executing.
animation-iteration-count	Specifies the number of times an animation cycle should be played before stopping.
animation-name	Specifies the name of @keyframes defined animations that should be applied to the selected element.

animation-play-state	Specifies whether the animation is running or paused.
animation-timing-function	Specifies how a CSS animation should progress over the duration of each cycle.
backface-visibility	Specifies whether or not the "back" side of a transformed element is visible when facing the user.
background	Defines a variety of background properties within one declaration.
background-attachment	Specify whether the background image is fixed in the viewport or scrolls.
background-clip	Specifies the painting area of the background.
background-color	Defines an element's background color.
background-image	Defines an element's background image.
background-origin	Specifies the positioning area of the background images.
background-position	Defines the origin of a background image.
background-repeat	Specify whether/how the background image is tiled.
background-size	Specifies the size of the background images.
border	Sets the width, style, and color for all four sides of an element's border.
border-bottom	Sets the width, style, and color of the bottom border of an element.
border-bottom-color	Sets the color of the bottom border of an element.
border-bottom-left-radius	Defines the shape of the bottom-left border corner of an element.
border-bottom-right-radius	Defines the shape of the bottom-right border corner of an element.
border-bottom-style	Sets the style of the bottom border of an element.
border-bottom-width	Sets the width of the bottom border of an element.
border-collapse	Specifies whether table cell borders are connected or separated.
border-color	Sets the color of the border on all the four sides of an element.
border-image	Specifies how an image is to be used in place of the border styles.
border-image-outset	Specifies the amount by which the border image area extends beyond the border box.
border-image-repeat	Specifies whether the image-border should be repeated, rounded or stretched.

border-image-slice	Specifies the inward offsets of the image-border.
border-image-source	Specifies the location of the image to be used as a border.
border-image-width	Specifies the width of the image-border.
border-left	Sets the width, style, and color of the left border of an element.
border-left-color	Sets the color of the left border of an element.
border-left-style	Sets the style of the left border of an element.
border-left-width	Sets the width of the left border of an element.
border-radius	Defines the shape of the border corners of an element.
border-right	Sets the width, style, and color of the right border of an element.
border-right-color	Sets the color of the right border of an element.
border-right-style	Sets the style of the right border of an element.
border-right-width	Sets the width of the right border of an element.
border-spacing	Sets the spacing between the borders of adjacent table cells.
border-style	Sets the style of the border on all the four sides of an element.
border-top	Sets the width, style, and color of the top border of an element.
border-top-color	Sets the color of the top border of an element.
border-top-left-radius	Defines the shape of the top-left border corner of an element.
border-top-right-radius	Defines the shape of the top-right border corner of an element.
border-top-style	Sets the style of the top border of an element.
border-top-width	Sets the width of the top border of an element.
border-width	Sets the width of the border on all the four sides of an element.
bottom	Specify the location of the bottom edge of the positioned element.
box-shadow	Applies one or more drop-shadows to the element's box.
box-sizing	Alter the default CSS box model.

caption-side	Specify the position of table's caption.
clear	Specifies the placement of an element in relation to floating elements.
clip	Defines the clipping region.
color	Specify the color of the text of an element.
column-count	Specifies the number of columns in a multi-column element.
column-fill	Specifies how columns will be filled.
column-gap	Specifies the gap between the columns in a multi-column element.
column-rule	Specifies a straight line, or "rule", to be drawn between each column in a multi-column element.
column-rule-color	Specifies the color of the rules drawn between columns in a multi-column layout.
column-rule-style	Specifies the style of the rule drawn between the columns in a multi-column layout.
column-rule-width	Specifies the width of the rule drawn between the columns in a multi-column layout.
column-span	Specifies how many columns an element spans across in a multi-column layout.
column-width	Specifies the optimal width of the columns in a multi-column element.
columns	A shorthand property for setting column-width and column-count properties.
content	Inserts generated content.
counter-increment	Increments one or more counter values.
counter-reset	Creates or resets one or more counters.
cursor	Specify the type of cursor.
direction	Define the text direction/writing direction.
display	Specifies how an element is displayed onscreen.
empty-cells	Show or hide borders and backgrounds of empty table cells.
flex	Specifies the components of a flexible length.
flex-basis	Specifies the initial main size of the flex item.
flex-direction	Specifies the direction of the flexible items.

flex-flow	A shorthand property for the flex-direction and the flex-wrap properties.
flex-grow	Specifies how the flex item will grow relative to the other items inside the flex container.
flex-shrink	Specifies how the flex item will shrink relative to the other items inside the flex container.
flex-wrap	Specifies whether the flexible items should wrap or not.
float	Specifies whether or not a box should float.
font	Defines a variety of font properties within one declaration.
font-family	Defines a list of fonts for element.
font-size	Defines the font size for the text.
font-size-adjust	Preserves the readability of text when font fallback occurs.
font-stretch	Selects a normal, condensed, or expanded face from a font.
font-style	Defines the font style for the text.
font-variant	Specify the font variant.
font-weight	Specify the font weight of the text.
height	Specify the height of an element.
justify-content	Specifies how flex items are aligned along the main axis of the flex container after any flexible lengths and auto margins have been resolved.
left	Specify the location of the left edge of the positioned element.
letter-spacing	Sets the extra spacing between letters.
line-height	Sets the height between lines of text.
list-style	Defines the display style for a list and list elements.
list-style-image	Specifies the image to be used as a list-item marker.
list-style-position	Specifies the position of the list-item marker.
list-style-type	Specifies the marker style for a list-item.
margin	Sets the margin on all four sides of the element.
margin-bottom	Sets the bottom margin of the element.

margin-left	Sets the left margin of the element.
margin-right	Sets the right margin of the element.
margin-top	Sets the top margin of the element.
max-height	Specify the maximum height of an element.
max-width	Specify the maximum width of an element.
min-height	Specify the minimum height of an element.
min-width	Specify the minimum width of an element.
opacity	Specifies the transparency of an element.
order	Specifies the order in which a flex items are displayed and laid out within a flex container.
outline	Sets the width, style, and color for all four sides of an element's outline.
outline-color	Sets the color of the outline.
outline-offset	Set the space between an outline and the border edge of an element.
outline-style	Sets a style for an outline.
outline-width	Sets the width of the outline.
overflow	Specifies the treatment of content that overflows the element's box.
overflow-x	Specifies the treatment of content that overflows the element's box horizontally.
overflow-y	Specifies the treatment of content that overflows the element's box vertically.
padding	Sets the padding on all four sides of the element.
padding-bottom	Sets the padding to the bottom side of an element.
padding-left	Sets the padding to the left side of an element.
padding-right	Sets the padding to the right side of an element.
padding-top	Sets the padding to the top side of an element.
page-break-after	Insert a page breaks after an element.
page-break-before	Insert a page breaks before an element.

page-break-inside	Insert a page breaks inside an element.
perspective	Defines the perspective from which all child elements of the object are viewed.
perspective-origin	Defines the origin (the vanishing point for the 3D space) for the perspective property.
position	Specifies how an element is positioned.
quotes	Specifies quotation marks for embedded quotations.
resize	Specifies whether or not an element is resizable by the user.
right	Specify the location of the right edge of the positioned element.
tab-size	Specifies the length of the tab character.
table-layout	Specifies a table layout algorithm.
text-align	Sets the horizontal alignment of inline content.
text-align-last	Specifies how the last line of a block or a line right before a forced line break is aligned when text-align is justify.
text-decoration	Specifies the decoration added to text.
text-decoration-color	Specifies the color of the text-decoration-line.
text-decoration-line	Specifies what kind of line decorations are added to the element.
text-decoration-style	Specifies the style of the lines specified by the text-decoration-line property
text-indent	Indent the first line of text.
text-justify	Specifies the justification method to use when the text-align property is set to justify.
text-overflow	Specifies how the text content will be displayed, when it overflows the block containers.
text-shadow	Applies one or more shadows to the text content of an element.
text-transform	Transforms the case of the text.
top	Specify the location of the top edge of the positioned element.
transform	Applies a 2D or 3D transformation to an element.
transform-origin	Defines the origin of transformation for an element.
transform-style	Specifies how nested elements are rendered in 3D space.

transition	Defines the transition between two states of an element.
transition-delay	Specifies when the transition effect will start.
transition-duration	Specifies the number of seconds or milliseconds a transition effect should take to complete.
transition-property	Specifies the names of the CSS properties to which a transition effect should be applied.
transition-timing-function	Specifies the speed curve of the transition effect.
vertical-align	Sets the vertical positioning of an element relative to the current text baseline.
visibility	Specifies whether or not an element is visible.
white-space	Specifies how white space inside the element is handled.
width	Specify the width of an element.
word-break	Specifies how to break lines within words.
word-spacing	Sets the spacing between words.
word-wrap	Specifies whether to break words when the content overflows the boundaries of its container.
z-index	Specifies a layering or stacking order for positioned elements.

CSS Pseudo-classes

A pseudo-class is used to define a special state of an element.
For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

```
/* unvisited link */
a:link {
  color: #FF0000;
}
/* visited link */
a:visited {
  color: #00FF00;
}
/* mouse over link */
a:hover {
  color: #FF00FF;
}
/* selected link */
a:active {
  color: #0000FF;
}
```

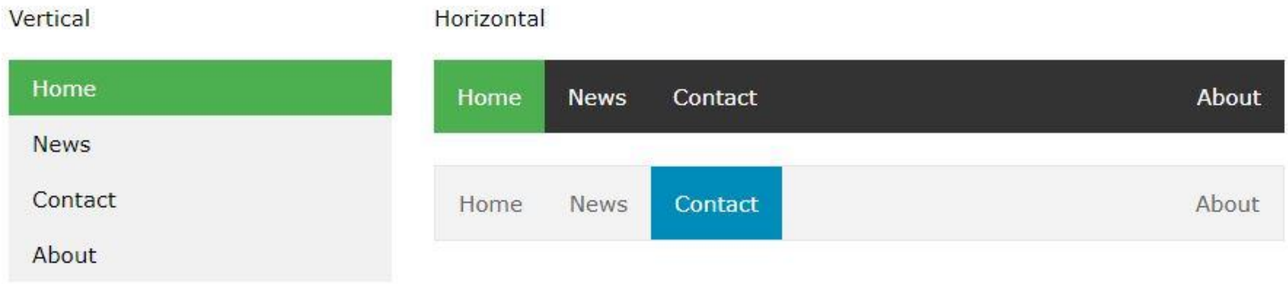
Note: a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective! a:active MUST come after a:hover in the CSS definition in order to be effective! Pseudo-class names are not case-sensitive.

CSS Navigation Bar

Having easy-to-use navigation is important for any web site.
With CSS you can transform boring HTML menus into good-looking navigation bars.

Navigation Bar = List of Links

A navigation bar needs standard HTML as a base.
In our examples we will build the navigation bar from a standard HTML list.
A navigation bar is basically a list of links, so using the <ul> and <li> elements makes perfect sense:

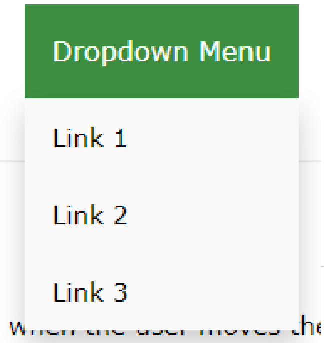


```
<ul>
  <li><a href="default.asp">Home</a></li>
  <li><a href="news.asp">News</a></li>
  <li><a href="contact.asp">Contact</a></li>
  <li><a href="about.asp">About</a></li>
</ul>


ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

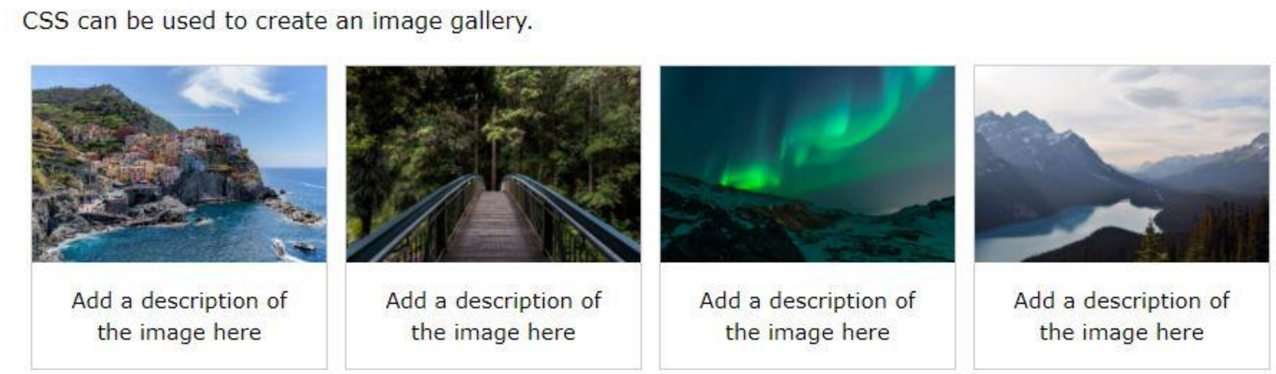
CSS Dropdowns

```
<style>
.dropdown {
  position: relative;
  display: inline-block;
}

.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  padding: 12px 16px;
  z-index: 1;
}
```

```
.dropdown:hover .dropdown-content {
  display: block;
}
</style>

<div class="dropdown">
  <span>Mouse over me</span>
  <div class="dropdown-content">
    <p>Hello World!</p>
  </div>
</div>
```



CSS Image Gallery



CSS can be used to create an image gallery.

Add a description of the image here

Add a description of the image here

Add a description of the image here

Add a description of the image here

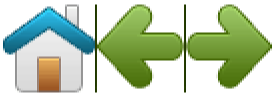
```
<html>
<head>
</head>
<body>

<div class="gallery">
  <a target="_blank" href="img_5terre.jpg">
    <img src="img_5terre.jpg" alt="Cinque
Terre" width="600" height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

<div class="gallery">
  <a target="_blank" href="img_forest.jpg">
    <img src="img_forest.jpg" alt="Forest" width="600" height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

<div class="gallery">
  <a target="_blank" href="img_lights.jpg">
    <img src="img_lights.jpg" alt="Northern
Lights" width="600" height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

<div class="gallery">
  <a target="_blank" href="img_mountains.jpg">
    <img src="img_mountains.jpg" alt="Mountains"
      width="600" height="400">
  </a>
  <div class="desc">Add a description
      of the image here</div>
</div>

</body>
</html>
```

```
<style>
div.gallery {
  margin: 5px;
  border: 1px solid #ccc;
  float: left;
  width: 180px;
}

div.gallery:hover {
  border: 1px solid #777;
}

div.gallery img {
  width: 100%;
  height: auto;
}

div.desc {
  padding: 15px;
  text-align: center;
}
</style>
```

CSS Image Sprites

An image sprite is a collection of images put into a single image.

A web page with many images can take a long time to load and generates multiple server requests. Using image sprites will reduce the number of server requests and save bandwidth.



```
#home {
  width: 46px;
  height: 44px;
  background: url(img_navsprites.gif) 0 0;
}

#next {
  width: 43px;
  height: 44px;
  background: url(img_navsprites.gif) -
91px 0;
}
```

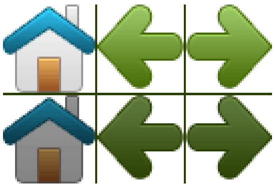
<img id="home" src="img_trans.gif"> - Only defines a small transparent image because the src attribute cannot be empty. The displayed image will be the background image we specify in CSS

width: 46px; height: 44px; - Defines the portion of the image we want to use

background: url(img_navsprites.gif) 0 0; - Defines the background image and its position (left 0px, top 0px)

This is the easiest way to use image sprites, now we want to expand it by using links and hover effects.

Image Sprites - Hover Effect



Our new image ("img_navsprites_hover.gif") contains three navigation images and three images to use for hover effects:

navigation images

Because this is one single image, and not six separate files, there will be no loading delay when a user hovers over the image.

We only add three lines of code to add the hover effect:

```
#home a:hover {
  background: url('img_navsprites_hover.gif') 0 -45px;
}

#prev a:hover {
  background: url('img_navsprites_hover.gif') -47px -45px;
}

#next a:hover {
  background: url('img_navsprites_hover.gif') -91px -45px;
}
```

CSS Attribute Selectors

The [attribute] selector is used to select elements with a specified attribute

The following example selects all <a> elements with a target attribute:

```
a[target] {
  background-color: yellow;
}
```

CSS [attribute="value"] Selector

The [attribute="value"] selector is used to select elements with a specified attribute and value.

The following example selects all <a> elements with a target="_blank" attribute:

```
a[target="_blank"] {
  background-color: yellow;
}
```

CSS [attribute~="value"] Selector

The [attribute~="value"] selector is used to select elements with an attribute value containing a specified word.

The following example selects all elements with a title attribute that contains a space-separated list of words, one of which is "flower":

```
[title~="flower"] {
  border: 5px solid yellow;
}
```

CSS [attribute|="value"] Selector

The [attribute|="value"] selector is used to select elements with the specified attribute starting with the specified value.

The following example selects all elements with a class attribute value that begins with "top":

Note: The value has to be a whole word, either alone, like class="top", or followed by a hyphen(-), like class="top-text"!

```
[class|="top"] {
  background: yellow;
}
```

CSS [attribute^="value"] Selector

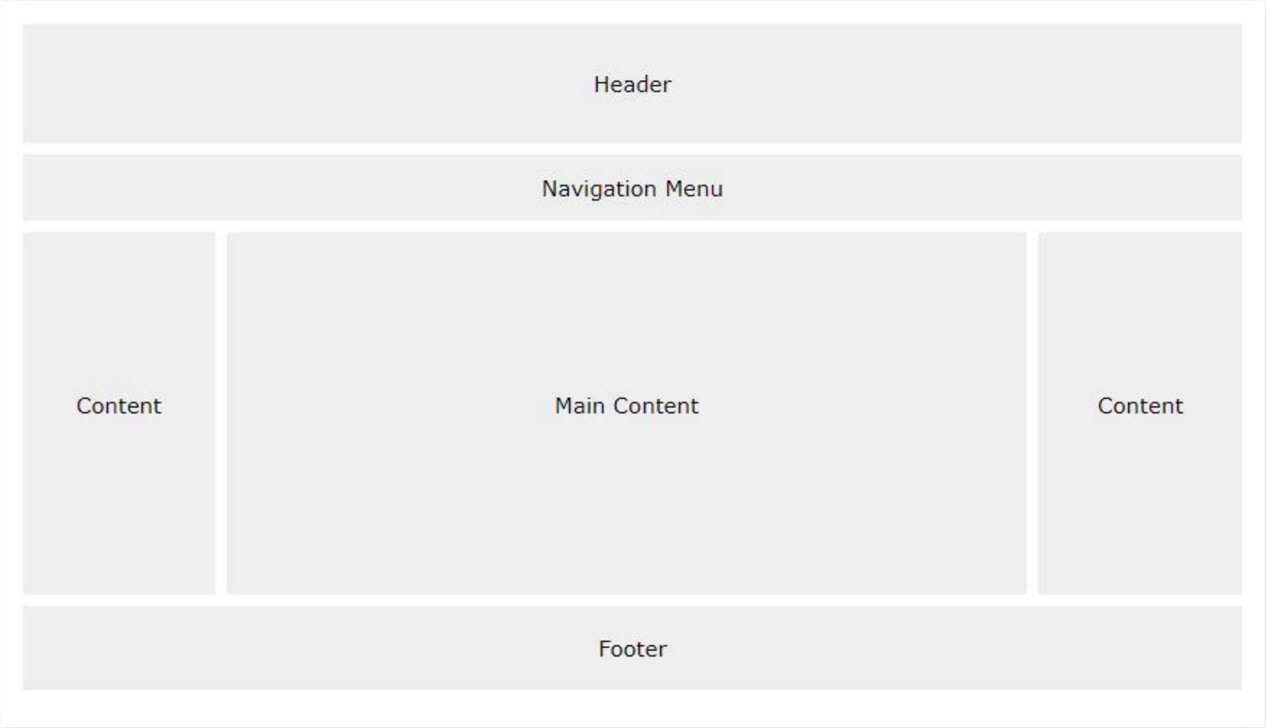
The [attribute^="value"] selector is used to select elements whose attribute value begins with a specified value.

The following example selects all elements with a class attribute value that begins with "top": Note: The value does not have to be a whole word!

```
[class^="top"] {
  background: yellow;
}
```

Website Layout

A website is often divided into headers, menus, content and a footer:



There are tons of different layout designs to choose from. However, the structure above, is one of the most common, and we will take a closer look at it in this tutorial.

Header

A header is usually located at the top of the website (or right below a top navigation menu). It often contains a logo or the website name:

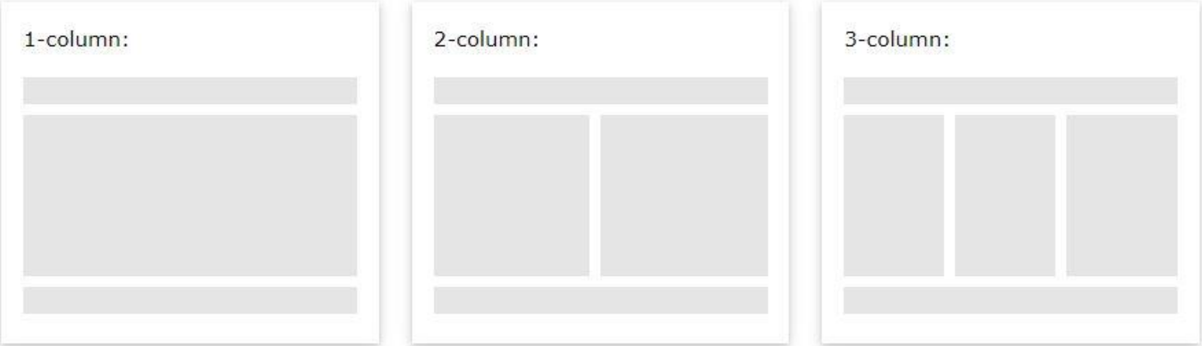
Navigation Bar

A navigation bar contains a list of links to help visitors navigating through your website:

Content

The layout in this section, often depends on the target users. The most common layout is one (or combining them) of the following:

- 1-column (often used for mobile browsers)
- 2-column (often used for tablets and laptops)
- 3-column layout (only used for desktops)



Footer
The footer is placed at the bottom of your page. It often contains information like copyright and contact info:

0.3 Java Scripts

JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved. It is the third layer of the layer cake of standard web technologies, two of which (HTML and CSS) we have covered in much more detail in other parts of the Learning Area.

the HTML code that will tell the browser that it needs to run a script. Once the browser sees these special tags, it interprets the JavaScript commands and will do what you have directed
it to do with your code. Thus, by simply editing an HTML document, you can begin using JavaScript on your Web pages and see the results.
For example, the following code adds some JavaScript to an HTML file that writes some text onto the Web page. Notice the addition of <script> and </script> tags. The code within them is JavaScript.

```
<html>
<body>
<script type="text/javascript">
document.write("This writes text to the page");
</script>
</body>
</html>
```

The next chapter looks at how to add JavaScript in an HTML file by using the <script> and </script> HTML tags. This will be your first step on the road to becoming a JavaScript coder

The <script> Tag
In HTML, JavaScript code is inserted between <script> and </script> tags

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript Functions and Events
A JavaScript function is a block of JavaScript code, that can be executed when "called" for.
For example, a function can be called when an event occurs, like when the user clicks a button.

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.
Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

JavaScript in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.
The function is invoked (called) when a button is clicked:

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page. The function is invoked (called) when a button is clicked:

```
<!DOCTYPE html>
<html>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display

External References

External scripts can be referenced with a full URL or with a path relative to the current web page. This example uses a full URL to link to a script:

```
<script src="https://iba.lk/js/myScript1.js"></script>
<script src="/js/myScript1.js"></script>
```

JavaScript Output

JavaScript can "display" data in different ways:

- Writing into an HTML element, using `innerHTML`.
- Writing into the HTML output using `document.write()`.
- Writing into an alert box, using `window.alert()`.
- Writing into the browser console, using `console.log()`.

Using innerHTML

To access an HTML element, JavaScript can use the document.getElementById(id) method. The id attribute defines the HTML element. The innerHTML property defines the HTML content:

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Using document.write()

For testing purposes, it is convenient to use document.write():

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>
</body>
</html>
```

Using console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

Using window.alert()

You can use an alert box to display data:

```
<script>
window.alert(5 + 6);
</script>
```

JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

The statements are executed, one by one, in the same order as they are written.

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

JavaScript Keywords

JavaScript statements often start with a keyword to identify the JavaScript action to be performed. Here is a list of some of the keywords you will learn about in this tutorial:

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

JavaScript Variables

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables, declared with the **var** keyword:

```
var x = 5;      x stores the value 5
var y = 6;      y stores the value 6
var z = x + y;  z stores the value 11
```

JavaScript Identifiers

All JavaScript variables must be identified with unique names.

These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

JavaScript Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:

(parameter1, parameter2, ...

The code to be executed, by the function, is placed inside curly brackets: {}

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

The code inside the function will execute when "something" invokes (calls) the function:

When an event occurs (when a user clicks a button)

When it is invoked (called) from JavaScript code

Automatically (self invoked)

Function Return

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

```
var x = myFunction(4, 3);   // Function is called, return value will
end up in x

function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Accessing a function without () will return the function object instead of the function result.

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

```
var x = toCelsius(77);
var text = "The temperature is " + x + " Celsius";
var text = "The temperature is " + toCelsius(77) + " Celsius";
```

JavaScript Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

In the following example, an onclick attribute (with code), is added to a <button> element:

```
<button onclick="document.getElementById('demo').innerHTML=Date()">
The time is?</button>
```

In the example above, the JavaScript code changes the content of the element with id="demo".

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more …

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

JavaScript Strings

A JavaScript string is zero or more characters written inside quotes.

```
var answer1 = "It's alright";
var answer2 = "He is called 'Johnny'";
var answer3 = 'He is called "Johnny"';
```

String Length

To find the length of a string, use the built-in length property

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Escape Character

Because strings must be written within quotes, JavaScript will misunderstand this string:

```
var x = "We are the so-called "Vikings" from the north.";
```

The string will be chopped to "We are the so-called ".
The solution to avoid this problem, is to use the backslash escape character.
The backslash (\) escape character turns special characters into string characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

```
var x = "We are the so-called \"Vikings\" from the north.";
```

Six other escape sequences are valid in JavaScript:

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line

\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

The 6 escape characters above were originally designed to control typewriters, teletypes, and fax machines. They do not make any sense in HTML.

Breaking Long Code Lines

For best readability, programmers often like to avoid code lines longer than 80 characters.
If a JavaScript statement does not fit on one line, the best place to break it is after an operator:
You can also break up a code line within a text string with a single backslash:

```
document.getElementById("demo").innerHTML = "Hello \
Dolly!";
```

JavaScript String Functions

Method	**Meaning**	**Example**
length	determines length of string	var a = “hackr.io”; a.length;
indexof()	finds position of the first occurrence of a character or text in the string	var a = “hackr.io is nice website”; var b = a.indexof(“nice”);
lastindexof()	returns last occurrence of text in a string	var a = “hackr.io is nice website”; var b = a.indexof(“nice”, 6);
search()	searches and returns position of a specified value in string	var a = “hackr.io is nice website”; var b = a.search(“nice”);
slice()	extracts and returns part of a string as another new string	var a = “hackr.io is nice website”; var b = a.slice(13); will return nice website.
substring()	substring returns part of the string from start index to the end index specified. cannot take negative values unlike slice()	var a = “hackr.io is nice website”; var b = a.substring(0, 7);
substr()	returns the sliced out portion of a string, the second parameter being the length of the final string.	var a = “hackr.io is nice website”; var b = a.substr(13, 8);
replace()	replaces a particular value with another	var a = “hackr.io is nice website”; var b = a.replace(“nice”, “good”);
touppercase()	changes all characters into uppercase	var a = “hackr.io is nice website”; var b = a.touppercase (a);
tolowercase()	changes all characters into lowercase	var a = “hackr.io is nice website”; var b = a.tolowercase(a);
concat()	joins two or more strings together into another string	var a = “my name is”; var b = “john”; var c = a.concat(“: ”, b);

trim()	removes white spaces from a string	var a = “hi, there! ”;a.trim();
charat()	finds character at a specified position	var a = “hackr.io”; a.charat(1) will return a
charcodeat()	returns the unicode of character at the specified position	“hackr”.charcodeat(0); will return 72
split()	convert a string into array based on special character	var a = “hackr.io”; var arr = a.split(“”); will return an array of characters h,a,c,k,r and so on..
accessing characters using []	access a character of string using its index (doesn’t work on some versions of ie)	var a = “hackr.io”; a[2] will return c

JavaScript Arrays

JavaScript arrays are used to store multiple values in a single variable.

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

```
var cars = ["Saab", "Volvo", "BMW"];
```

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

```
var cars = ["Saab", "Volvo", "BMW"];
var cars = new Array("Saab", "Volvo", "BMW");
```

Access the Elements of an Array

You access an array element by referring to the index number

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Changing an Array Element

This statement changes the value of the first element in cars:

Functions	**Description**
concat()	Concatenate different arrays into one.
join()	Joins all the elements of one array as a string
indexof()	Returns the index (first position) of an element in the array
lastindexof()	Returns the last position of an element in the array
sort()	Alphabetic sort of array elements
reverse()	Sort elements in descending order
valueof()	Primitive value of the element specified
slice()	Cut a portion of one array and put it in a new array
splice()	Add elements to an array in a specific manner and position
unshift()	Add new element to the array in the beginning
shift()	Remove first element of the array
pop()	Remove the last element of the array
push()	Add new element to the array as the last one
tostring()	Prints the string value of the elements of the array

JavaScript Array Iteration Methods

Array.forEach()

The forEach() method calls a function (a callback function) once for each array element.

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt = txt + value + "<br>";
}
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value) {
  txt = txt + value + "<br>";
}
```

JavaScript Date Output

By default, JavaScript will use the browser's time zone and display a date as a full text string:

Fri Jan 01 2021 23:07:18 GMT+0530 (India Standard Time)

Creating Date Objects

Date objects are created with the new Date() constructor.

There are 4 ways to create a new date object:

```
new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
```

Date()	**Creates a new date object with current date and time**
Date(2019, 10, 21, 12, 24, 58, 13)	Create a custom date object. Format – (yyyy, mm, dd, hh, min, s, ms). Except for year and month, all parameters are optional.
Date("2019-10-21")	Date declaration as a string
getDate()	Get the day of the month as a number (1-31)
getDay()	The weekday as a number (0-6)
getFullYear()	Year as a four-digit number (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the millisecond (0-999)
getMinutes()	Get the minute (0-59)
getMonth()	Month as a number (0-11)
getSeconds()	Get the second (0-59)
getTime()	Get the milliseconds since January 1, 1970
getUTCDate()	The day (date) of the month in the specified date according to universal time (also available for day, month, full year, hours, minutes etc.)
parse	Parses a string representation of a date and returns the number
setDate()	Set the day as a number (1-31)
setFullYear()	Sets the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set milliseconds (0-999)
setMinutes()	Sets the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Sets the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)
setUTCDate()	Sets the day of the month for a specified date according to universal time (also available for day, month, full year, hours, minutes etc.)

JavaScript if else and else if

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

JavaScript Switch Statement

Use the switch statement to select one of many code blocks to be executed.

This is how it works:

1. The switch expression is evaluated once.
2. The value of the expression is compared with the values of each case.
3. If there is a match, the associated block of code is executed.
4. If there is no match, the default code block is executed.

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  }
```

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value. Often this is the case when working with arrays:

for	looping in javascript	var i;
		for (i = 0; i < 5; i++
		{ // code}

while	execute a block of code while some condition is true	while (product.length > 5)
		{// some code}

do… while	similar to while, but executes at least as the condition is applied after the code is executed	do {
		// code
		}while (condition){
		}

break	break and exit the cycle based on some conditions	if (i <10)
		break;

continue	continue next iteration if some conditions are met	if (j>10)
		continue;

JavaScript Regular Expressions

A regular expression is a sequence of characters that forms a search pattern.

When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of text search and text replace operations.

Using String Methods

In JavaScript, regular expressions are often used with the two string methods: search() and replace().

The search() method uses an expression to search for a match, and returns the position of the match.

The replace() method returns a modified string where the pattern is replaced.

Using String search() With a String

The search() method searches a string for a specified value and returns the position of the match:

```
var str = "Visit IBA Campus";
var n = str.search(/IBA/i);
```

e	evaluate replacement
i	case-insensitive matching
g	global matching – find all matches
m	multiple line matching
s	treat strings as a single line
x	allow comments and whitespace in the pattern
u	ungreedy pattern

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

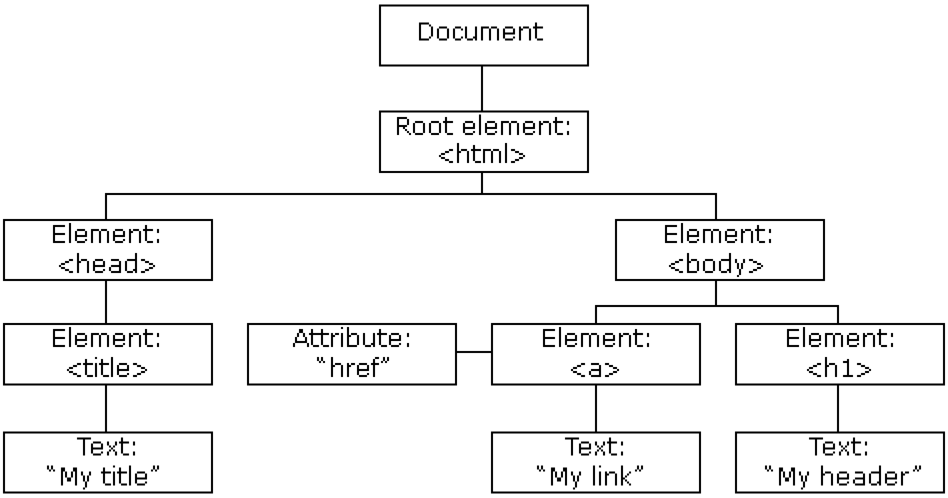
```
<form name="myForm" action="/action_page.php" onsubmit="return
validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>


function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

JavaScript HTML DOM

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a Document Object Model of the page.

The HTML DOM model is constructed as a tree of Objects:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	**Description**
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

Changing HTML Elements

Property	**Description**
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element

element.style.property = new style	Change the style of an HTML element
Method	Description
element.setAttribute(attribute, value)	Change the attribute value of an HTML element

Adding and Deleting Elements

Method	**Description**
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

Adding Events Handlers

Method	**Description**
document.getElementById(id).onclick = function(){code}	Adding event handler code to an onclick event

Finding HTML Objects

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

Later, in HTML DOM Level 3, more objects, collections, and properties were added.

Property	**Description**	**DOM**
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete. Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1

document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

JSON - Introduction

- JSON: JavaScript Object Notation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.

Exchanging Data

When exchanging data between a browser and a server, the data can only be text.

JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.

We can also convert any JSON received from the server into JavaScript objects.

This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

Why use JSON?

Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

JSON.parse()

So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

Sending Data

If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

```
var myObj = {name: "John", age: 31, city: "New York"};
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

Receiving Data

If you receive data in JSON format, you can convert it into a JavaScript object:

```
var myJSON = '{"name":"John", "age":31, "city":"New York"}';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

Valid Data Types

In JSON, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- **null**

JSON Strings

Strings in JSON must be written in double quotes.

```
{ "name":"John" }
```

JSON Numbers

Numbers in JSON must be an integer or a floating point.

```
{ "age":30 }
```

JSON Objects

Values in JSON can be objects.

```
{
"employee":{ "name":"John", "age":30, "city":"New York" }
}
```

JSON Arrays

Values in JSON can be arrays.

```
{
"employees":[ "John", "Anna", "Peter" ]
}
```

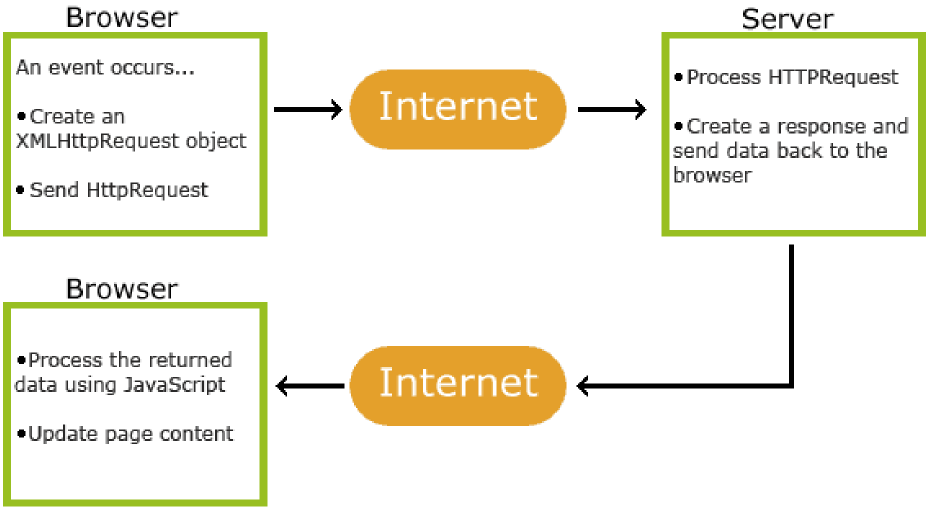
AJAX Introduction

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.
AJAX is not a programming language.
AJAX just uses a combination of:

- A browser built-in `XMLHttpRequest` object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

The XMLHttpRequest object can be used to exchange data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Create an XMLHttpRequest Object

All modern browsers (Chrome, Firefox, IE7+, Edge, Safari, Opera) have a built-in XMLHttpRequest object. Syntax for creating an XMLHttpRequest object:

```
variable = new XMLHttpRequest();
```

XMLHttpRequest Object Methods

Method	**Description**
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(method, url, async, user, psw)	Specifies the request method: the request type GET or POST url: the file location async: true (asynchronous) or false (synchronous) user: optional user name psw: optional password
send()	Sends the request to the server Used for GET requests
send(string)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

XMLHttpRequest Object Properties

Property	**Description**
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received

	3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
      xhttp.open("GET", "ajax_info.txt", true);
      xhttp.send();
```

Method	Description
open(method, url, async)	Specifies the type of request method: the type of request: GET or POST url: the server (file) location async: true (asynchronous) or false (synchronous)
send()	Sends the request to the server (used for GET)
send(string)	Sends the request to the server (used for POST)

GET Requests

GET request:

```
      xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);
      xhttp.send();
```

POST Requests

A simple POST request:

```
      xhttp.open("POST", "ajax_test.asp", true);
      xhttp.setRequestHeader("Content-type", "application/x-www-form-
      urlencoded");
      xhttp.send("fname=Henry&lname=Ford");
```

AJAX - Server Response

he readyState property holds the status of the XMLHttpRequest.

The onreadystatechange property defines a function to be executed when the readyState changes.

The status property and the statusText property holds the status of the XMLHttpRequest object.

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The `onreadystatechange` function is called every time the readyState changes.

When `readyState` is 4 and status is 200, the response is ready:

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

PHP Introduction

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use
- PHP is an amazing and popular language!

It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)!
It is deep enough to run the largest social network (Facebook)!
It is also easy enough to be a beginner's first server side language!

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code is executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"
- What Can PHP Do?
- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies

- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side
- What's new in PHP 7
- PHP 7 is much faster than the previous popular stable release (PHP 5.6)
- PHP 7 has improved Error Handling
- PHP 7 supports stricter Type Declarations for function arguments
- PHP 7 supports new operators (like the spaceship operator: <=>)

Setup and Installation

Since PHP is a server-side technology, you should naturally expect to invest some time in setting up a server environment for production, development or learning. To be frank, PHP is quite easy to set up compared to other monsters like J2EE. Nevertheless, the procedures are complicated by the various combinations of different versions of web server, PHP and database (most often MySQL). Below I will introduce the steps needed to set up a working PHP environment with MySQL database.

Linux

- If your desktop runs on Linux, chances are that Apache, PHP, and MySQL are already installed for you. This wildly popular configuration is commonly referred to as LAMP, i.e. Linux Apache MySQL PHP, or P, the latter 'P', can also refer to Perl another major player in the opensource web service arena. If some components are not installed, you will likely have to manually install the following packages:

Apache or Lighttpd
PHP
MySQL or Postgres
The PHP integration plugin for the database.

Debian or its derivatives

On Debian or its derivatives, Ubuntu included[1], you can use the corresponding commands:

```
apt-get install php5

## Server
#### If you wish to use Apache apt-get install apache2
## -or-
#### If you wish to use Lighttpd apt-get install lighttpd

## Database
#### If you wish to use Postgres
apt-get install postgres-server    postgres-client    php5-pg
## -or-
```

```
#### If you wish to use Mysql
apt-get install mysql-server      mysql-client      php5-mysql
```

^ If you chose to use Ubuntu with Apache and MySQL you might wish to utilize the Ubuntu community site for such a configuration ubuntu lamp wiki.

Gentoo

For Gentoo Linux users, the gentoo-wiki has this HowTo available: Apache2 with PHP and MySQL.

In general, you'll want to do the following under Gentoo:

```
emerge apache emerge mysql emerge mod_php
```

RPM-based

The exact procedures depend on your Linux distribution. On a Fedora system, the commands are typically as follows:

```
yum install httpd yum install php yum install mysql
yum install php-mysql
```

It's impossible to cover all the variants here, so consult your Linux distribution's manual for more details, or grab a friend to do it for you.

One sure-fire way of getting PHP up and running on your *nix system is to compile it from source. This isn't as hard as it may sound and there are good instructions available in the PHP manual.

Windows

Contrary to what some people may think, PHP on Windows is a very popular option. On a Windows platform, you have the option to use either the open source Apache web server, or the native Internet Information Services (IIS) server from Microsoft, which can be installed from your Windows CD. When you have one of these servers installed, you can download and install the appropriate PHP Windows binaries distributions from PHP download page. The installer version requires less user-interaction.

For increased performance you will want to use FastCGI. There is a wikibook that will assist you on Setting up IIS with FastCGI.

Databases

On Microsoft Windows you must always install your own database. Two popular choices are the open source Postgres, and MySQL. Postgres is more liberally licensed, and is free to use for commercial purposes.

Postgresql

Official Zend documentation: http://us.php.net/pgsql

Postgres is simple and easy to install, browse to http://www.postgresql.org/ftp/binary/v8.3.0/win32/ and download the exe and double-click.

MySQL

Official MySQL documentation: http://us.php.net/mysql

You might wish to install the MySQL database. You can download the Windows version of MySQL, and follow the installation instructions. If you have PHP 4, you do not need to install the equivalence of php-mysql on Linux, as MySQL support is built-in in Windows distributions of PHP. In PHP 5 you will need to uncomment the following line in your php.ini file (that is, remove the ';' at the beginning of the line):

```
;extension=php_mysql.dll
```

Bundled Package

If you find all the above too much a hassle, you have another option. Driven by the eternal desire to do things the safe/easy way, several conveniently packaged AMP bundles of Apache/MySQL/PHP can be found on the net. One of them is PHPTriad. Or, you can try Uniform Server. It is a small WAMP Package. 1 click install and also easy to use. Same with XAMPP for Windows. And after trying these out you can simply delete the directory and everything is clean. A number of portable Windows AMP package choices are summarized at List of portable Web Servers.

Also, a package installer called WAMP is available. It simply installs Apache, PHP and MySQL on windows with ease. http://www.en.wampserver.com/

Mac OS X

Mac OS X comes with Apache server as standard, and enabling it is as simple as checking the box next to 'Personal Web Sharing' in the 'Sharing' section of System Preferences. Once you have done this you can place files in /Library/WebServer/Documents to access them on your server. Mac OS X does come with PHP but the installation lacks any significant quantity of extensions, so if you want any you're going to have to install PHP yourself. You can do this by following the instructions in Apple's Developer Connection, or you can download an automatic installer such as the ones available at Entropy. Once you've done one of those, you'll have a server with PHP running on your Mac.

To install MySQL just download and run the OS X installer package or use XAMPP for MacOS X.

If you use unix or learning it, however, compiling might be the way to go for all three, or just the ones you like. The advantage is that you can choose exactly which extensions you want for PHP and Apache. Also you can choose which versions to compile together. To do this make sure you have the Developer Tools installed. They are shipped with OS X.

How Do I Know My Setup is Working?

After you have successfully completed the previous section, it's time to make sure that everything went well. You also get the chance to write your very first PHP scripts! Open your favourite plain text editor (not Microsoft Word or another word processor), and type the following magical line:

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

Save it as phpinfo.php in your web server's root document directory. If you are using a web hosting server, upload it to the server to where you would place HTML files. Now, open up your web browser, and go to http://localhost/phpinfo.php, or http://your-web-hosting- server.com/phpinfo.php if you are using a web hosting server, and look at the output.

Now scroll down that page and make sure there is a table with the title "mysql", and the top row should read: "MySQL support: enabled". If your output does not have this, your particular installation of PHP does not have MySQL support enabled. Note that this test doesn't tell you whether MySQL server is running. You should fire up your MySQL client and check before you proceed.

Some dedicated php or script editors even have color coding of different words which can be very useful for finding mistakes. A free implementation of this is the powerful Notepad++, available from Sourceforge and licensed under the GPL.

Hello World

"Hello world." is the first program most beginning programmers will learn to write in any given language. Here is an example of how to print "Hello world!" in PHP.

Code:

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

Output: Hello world!

This is as basic as PHP gets. Three simple lines, the first line identifies that everything beyond the `<?php tag, until the ?>` tag, is PHP code. The second line causes the greeting to be printed (or echoed) to the web page. This next example is slightly more complex and uses variables.

Hello World With Variables

This example stores the string "Hello world!" in a variable called $string. The following lines show various ways to display the variable $string to the screen.

PHP Code:

```
<?php

// Declare the variable 'string' and assign it a value.
// The <br /> is the HTML equivalent to a new line.
$string = 'Hello world!<br />';

// You can echo the variable, similar to the way you would echo a string.
echo $string;

// You could also use print. print $string;

// Or, if you are familiar with C, printf can be used too. printf('%s',
$string);
?>
```

PHP Output:

```
Hello world!<br />Hello world!<br />Hello world!<br />
```

The previous example contained two outputs. PHP can output HTML that your browser will format and display. The PHP Output box is the exact PHP output. The HTML Render box is approximately how your browser would display that output. Don't let this confuse you, this is just to let you know that PHP can output HTML. We will cover this much more in depth later.

Variables

Variables are the basis of any programming language: they are "containers" (spaces in memory) which hold data. The data can change, thus it is "variable".

If you've had any experience with other programming languages, you know that in some of the languages, you must define the type of data that the variable will hold. Those languages are called statically-typed, because the types of variables must be known before you store something in them. Programming languages such as C++ and Java are statically-typed. PHP, on the other hand, is dynamically-typed, because the type of the variable is linked to the value of the variable. You could define a variable for a string, store a string, and then replace the string with a number. To do the same thing in C++, you would have to cast, or change the type of, the variable, and store it in a different "container".

All variables in PHP follow the format of a dollar sign ($) followed by an identifier i.e. $variable_name. These identifiers are case-sensitive, meaning that capitalization matters, so $wiki is different from $Wiki.

Real world analogy

To compare a variable to real world objects, imagine your computer's memory as a storage shed. A variable would be a box in that storage shed and the contents of the box (such as a cup) would be the data in that variable.

If the box was labeled kitchen stuff and the box's contents were a cup, the PHP code would be:

```
$kitchen_stuff = 'cup';
```

If I then went into the storage shed, opened the box labeled kitchen stuff, and then replaced the cup with a fork, the new code would be:

```
$kitchen_stuff = 'fork';
```

Notice the addition of the = in the middle and the ; at the end of the code block. The = is the assignment operator, or in our analogy, instructions that came with the box that states "put the cup in the box". The ; indicates to stop evaluating the block of code, or in our analogy, finish up with what you are doing and move on to something else.

Also notice the cup was wrapped in single quotes instead of double. Using double quotes would tell the PHP parser that there may be more than just a cup going into the box and to look for additional instructions.

```
      $bathroom_stuff = 'toothbrush';
      $kitchen_stuff = "cup $bathroom_stuff";
      //$kitchen_stuff contents is now ‘cup toothbrush'
```

Single quotes tell the PHP parser that it's only a cup and to not look for anything more. In this example the bathroom box that should've had its contents added to the kitchen box has its name added instead.

```
      $bathroom_stuff = 'toothbrush';
      $kitchen_stuff = 'cup $bathroom_stuff';
      //$kitchen_stuff contents is now '''cup $bathroom_stuff'''
```

So again, try to visualize and associate the analogy to grasp the concept of variables with the comparison below. Note that this is a real world object comparison and NOT PHP code.

Computer memory (RAM) = storage shed
Variable = a box to hold stuff
Variable name = a label on the box such as kitchen stuff
Variable data = the contents of the box such as a cup

Notice that you wouldn't name the variable box, as the relationship between the variable and the box is represented by the code>$ and how the data is stored in memory. For example a constant and array can be considered a type of variable when using the box analogy as they all are containers to hold some sort of contents, however, the difference is on how they are defined to handle the contents in the box.

Variable: a box that can be opened while in the storage shed to exchange the contents in the box.

Constant: a box that cannot be opened to exchange its contents. Its contents can only be viewed and not exchanged while inside the storage shed.

Array: a box that contains 1 or more additional boxes in the main box. To complicate matters for beginners, each additional box may contain a box as well. In the kitchen stuff box we have two boxes, the clean cup box

```
$kitchen_stuff["clean_cup"] = 'the clean cup';
and the dirty cup box
$kitchen_stuff["dirty_cup"] = 'the dirty cup'; More on variables, from the PHP
manual The print and echo statements
```

Print is the key to output. It sends whatever is in the quotes (or parentheses) which follow it to the output device (browser window). A similar function is echo, but print allows the user to check whether or not the print succeeded.
When used with quotation marks, as in:
print "Hello, World!";

The quoted text is treated as if it were a string, and thus can be used in conjunction with the concatenation (joining two strings together) operator as well as any function that returns a string value.

The following two examples have the same output. print "Hello, World!";
print "Hello" . ", " . "World!";

The dot symbol concatenates two strings. In other programming languages, concatenating a string is done with the plus symbol and the dot symbol is generally used to call functions from classes.

Also, it might be useful to note that under most conditions echo can be used interchangably with print. print returns a value, so it can be used to test if the print succeeded, while echo assumes everything worked. Under most conditions there is nothing we can do if echo fails.
The following examples have the same output again. echo "Hello, World!";

and
echo "Hello" . ", " . "World!";

We will use echo in most sections of this book, since it is the more commonly used statement.

It should be noted that while echo and print can be called in the same way as functions, they are, in fact, language constructs, and can be called without the brackets. Normal functions (almost all others) must be called with brackets following the function identifier.

BASICS

The Examples

Example 1 - Basic arithmetic operators

This example makes use of the five basic operators used in mathematical expressions. These are the foundation of all mathematical and string operations performed in PHP.

The five mathematical operators all function identically to those found in C++ and Java add (+) subtract (-) multiply (*) divide (/) assign (=)

Examine this example. Each mathematical expression to the right of the assign operator is evaluated, using the normal order of operations. When the expression has been
evaluated, the resultant value is assigned to the variable named to the left of the assign operator.

PHP Code:

```
<?php
$x = 25;
$y = 10;
$z = $x + $y;
echo $z;

echo "<br />";
$z = $x / $y;
echo $z;

echo "<br />";
$z = $y * $y * $x; echo $z - 1250; echo "<br />";
?>
```

PHP Output:

```
35<br />2.5<br />1250<br />
```

HTML Render:

```
35
2.5
1250
```

Note: If you are not familiar with (X)HTML, you may not know the purpose of this part of the above code:

```
echo "<br />";
```

Its purpose is to insert an HTML "line break" between the results, causing the browser to display each result on a new line when rendering the page. In the absence of this line, the above code would instead print: 352.51250

This is of course not the desired result.

There are two code options which perform the opposite of the assign (=) operator. The keyword null should be used for variable nullification, which is actually used with the assign (=) operator in place of a value. If you want to destroy a variable, the unset() language construct is available.

Examples:
$variable = null;
or unset($variable);

Example 2 - String concatenation

This example demonstrates the concatenation operator (.), which joins together two strings, producing one string consisting of both parts. It is analogous to the plus (+) operator commonly found in C++ string class (see STL), Python, Java, JavaScript implementations.

The statement
$string = $string . " " . "All the cool kids are doing it.";
prepends the current value of $string (which is "PHP is wonderful and great.") to the literal string " All the cool kids are doing it." and assigns this new string to $string.
Code:

```
<?php
$string = "PHP is wonderful and great.";
$string = $string . " " . "All the cool kids are doing it.";
echo $string;
?>
```

Output:
PHP is wonderful and great. All the cool kids are doing it. Example 3 - Shortcut operators
This snippet demonstrates self-referential shortcut operators. The first such operator is the
++ operator, which increments $x (using the postfix form) by 1 giving it the value 2. After incrementing $x, $y is defined and assigned the value 5.
The second shortcut operator is *=, which takes $y and assigns it the value $y * $x, or 10. After initializing $z to 180, the subsequent line performs two shortcut operations. Going by
order of operations (see manual page below), $y is decremented (using the prefix form)
and divided into $z. $z is assigned to the resulting value, 20. Code:

```
<?php
$x = 1;
$x++;
echo $x . " ";
$y = 5;
$y *= $x;
echo $y . " ";
$z = 180;
$z /= --$y;
echo $z;
?>
```

Output:
2 10 20

Note: The expanded version of the above code (without the shortcut operators) looks like this:

```
<?php
$x = 1;
$x = $x + 1;
echo $x . " ";
$y = 5;
$y = $y * $x;
echo $y . " ";
$z = 180;
$y = $y - 1;
$z = $z / $y;
echo $z;
?>
```

The output is the same as seen in the above example.

Operators

An operator is any symbol used in an expression used to manipulate data. The seven basic PHP operators are:

= (assignment)
+ (addition)
- (subtraction)
* (multiplication)
/ (division)
% (modulus)
. (concatenation)

In addition, each of the above operators can be combined with an assignment operation, creating the operators below:

+= (addition assignment)
-= (subtraction assignment)
*= (multiplication assignment)
/= (division assignment)
%= (modulus assignment)
.= (concatenation assignment)

These operators are used when a variable is added, subtracted, multiplied or divided by a second value and subsequently assigned to itself.
In other words, the statements
$var = $var + 5;
and
$var += 5;
are equivalent.

There are also increment and decrement operators in PHP.
++ (increment)
-- (decrement)

These are a special case of the addition and subtraction assignment operators.
This code uses the addition assignment operator to increment and decrement a variable.

Code:

```
$var = 0;

$var += 1;
echo "The incremented value is $var.\n";

$var -= 1;
echo "The decremented value is $var.\n";
```

Output:

```
The incremented value is 1. The decremented value is 0.
```

While this is perfectly legal in PHP, it is somewhat lengthy for an operation as common as this. It can easily be replaced by the increment operator, shortening the statement.
This code snippet uses the increment and decrement operators to increase and decrease a variable's value by one.

Code:

```
$var = 3;

$var++;
echo "The incremented value is $var.\n";

$var--;
echo "The decremented value is $var.\n";
```

Output:

```
The incremented value is 4. The decremented value is 3.
```

Using the increment operator makes your code slightly easier to read and understand.

For a more in-depth overview of PHP's operators, including an explanation of bitwise operators, refer to the manual link below.

Newline and Other Special Characters

Both of the below examples make use of the newline character (\n) to signify the end of the current line and the beginning of a new one.

The newline is used as follows: Code:

```
echo "PHP is cool,\nawesome,\nand great.";
```

Output: PHP is cool, awesome, and great.

Notice: the line break occurs in the output wherever the \n occurs in the string in the echo statement. However, a \n does not produce a newline when the HTML document is displayed in a web browser. This is because the PHP engine does not render the script. Instead, the PHP engine outputs HTML code, which is subsequently rendered by the web browser. The linebreak \n in the PHP script becomes HTML whitespace, which is skipped when the web browser renders it (much like the whitespace in a PHP script is skipped when the PHP engine generates HTML). This does not mean that the \n operator is useless; it can be used to add whitespace to your HTML, so if someone views the HTML generated by your PHP script they'll have an easier time reading it.

In order to insert a line-break that will be rendered by a web browser, you must instead use the <br /> tag to break a line.

Therefore the statement above would be altered like so:

```
echo 'PHP is cool,<br />awesome<br />and great.';
```

The function nl2br() is available to automatically convert newlines in a string to <br /> tags. The string must be passed through the function, and then reassigned:

PHP Code:

```
$string = "This\ntext\nbreaks\nlines.";
$string = nl2br($string);
print $string;
```

PHP Output: This<br /> text<br /> breaks<br /> lines.
HTML Render: This
text breaks lines.
Additionally, the PHP output (HTML source code) generated by the above example includes linebreaks.

Other special characters include the ASCII NUL (\0) - used for padding binary files, tab (\t) - used to display a standard tab, and return (\r) - signifying a carriage return. Again, these characters do not change the rendering of your HTML since they add whitespace to the HTML source. In order to have tabs and carriage returns rendered in the final web page, &tab; should be used for tabs and <br /> should be used for a carriage return.

Input to PHP

PHP has a set of functions that retrieve input. If you are using stanard input (such as that from a command-line), it is retrieves using the basic input functions:
Reading from standard input:

```
$mystring = fgets($stdin);
```

Or:

```
$stdin = fopen('php://stdin', 'r'); // opens standard input
$line = fgets($stdin); // reads until user presses ENTER
```

Webservers

On webservers, information sent to a PHP app may either be a GET operation or a POST operation.

For a GET operation, the parameters are sent through the address bar. Parameters within the bar may be retrieves by using accessing $_GET['parameter']. On a POST operation, submitted input is accessed by $_POST['parameter'].

A more generic array, $_REQUEST['parameter'] contains the contents of both $_GET, $_POST, and $_COOKIE.

Comments

As you write more complex scripts, you'll see that you must make it clear to yourself and to others exactly what you're doing and why you're doing it. Comments and "good" naming can help you make clear and understandable scripts because:

When writing a script takes longer than a week, by the time you're done, you won't remember what you did when you started, and you will most likely need to know.

Any script that is commonly used will need rewriting sooner or later. Rewriting is much easier (and in many cases, made possible) when you write down what you did.

If you want to show someone your scripts, they should be nice and neat.

Comments are pieces of code that the PHP parser skips. When the parser spots a comment, it simply keeps going until the end of the comment without doing anything. PHP offers both one line and multi-line comments.

One-Line Comments

One-line comments are comments that start where ever you start them and end at the end of the line. With PHP, you can use both // and # for your one-line comments (# is not commonly used). Those are used mainly to tell the reader what you're doing the next few lines. For example:

```
//Print the variable $message echo $message;
```

It's important to understand that a one-line comment doesn't have to 'black out' the whole line, it starts where ever you start it. So it can also be used to tell the reader what a certain variable does:

```
$message = ""; //This sets the variable $message to an empty string
```

The $message = ""; is executed, but the rest of the line is not.

Multi-Line Comments

This kind of comment can go over as many lines as you'd like, and can be used to state what a function or a class does, or just to contain comments too big for one line. To mark the beginning of a multiline comment, use /* and to end it, use */ . For example:

```
/* This is a
multiline comment And it will close When I tell it to.
*/
```

You can also use this style of comment to skip over part of a line. For example:

```
$message = ""/*this would not be executed*/;
```

Although it is recommended that one does not use this coding style, as it can be confusing in some editors.

Naming

Naming your variables, functions and classes correctly is very important. If you define the following variables:

```
$var1 = "PHP";
$var2 = 15;
```

They won't say much to anyone. But if you do it like this:

```
$programming_language = "PHP";
$menu_items = 15;
```

It would be much clearer. But don't go too far. $programming_language, for example is not a good name. It's too long, and will take you a lot of time to type and can affect clarity. A better name may be $prog_language, because it's shorter but still understandable. Don't forget to use comments as well, to mark what every variable does.

```
$prog_language = "PHP";  //The programming language used to write this script
$menu_items = 15;        //The maximum number of items allowed in your personal menu
```

Magic numbers

When using numbers in a program it is important that they have a clear meaning. For instance it's better to define $menu_items early on instead of using 15 repeatedly without telling people what it stands for. The major exception to this is the number 1; often programmers have to add or subtract 1 from some other number to avoid off-by-one errors, so 1 can be used without definition.

When you define numbers early on in their usage it also makes it easier to adjust the values later. Again if we have 15 menu items and we refer to them ten times, it will be a lot easier to adjust the program when we add a 16th menu item; just correct the variable definition and you have updated the code in 10 places.

Spacing

PHP ignores extra spaces and lines. This means, that even though you could write code like this:

```
if($var == 1) {echo "Good";} else {echo "Bad";}

It's better like this: if($var == 1) { echo "Good";
} else {
echo "Bad";
}

Some programmers prefer this way of writing:
if($var == 1)
{
echo "Good";
}
else
{
echo "Bad";
}
```

You should also use blank lines between different portions of your script. Instead of

```
$var = 1;
echo "Welcome!&lt;br /&gt;";
echo "How are you today?&lt;br /&gt;";
echo "The answer: ";
if($var == 1) {
echo "Good";
} else {
echo "Bad";
}
```

You could write:

```
      $var = 1;

      echo "Welcome!&lt;br /&gt;";
      echo "How are you today?&lt;br /&gt;";

      echo "The answer: ";
      if($var == 1) {
      echo "Good";
      } else {
      echo "Bad";
      }
```

And the reader will understand that your script first declares a variable, then welcomes the user, and then checks the variable.

Arrays

Arrays are sets of data which can be defined in a PHP Script. Arrays can contain other arrays inside of them without any restriction (hence building multidimensional arrays). Arrays can be referred to as tables or hashes.

Arrays can be created in two ways. The first involves using the function array. The second involves using square brackets.

The array function method

In the array function method, you create a array in the scheme of:
$foo = bar()

For example, to set the array up to make the keys sequential numbers (Example: "0, 1, 2, 3"), you use:
$foobar = array($foo, $bar);

This would produce the array like this:
$foobar[0] = $foo
$foobar[1] = $bar

It is also possible to define the key value:
$foobar = array('foo' => $foo, 'bar' => $bar);

This would set the array like this:
$foobar['foo'] = $foo
$foobar['bar'] = $bar

The square brackets method

The square brackets method allows you to set up by directly setting the values. For example, to make $foobar[1] = $foo, all you need to do is:
$foobar[1] = $foo;

The same applies for setting the key value:
$foobar['foo'] = $foo; Examples of Arrays
Example #1
This example sets and prints arrays.

PHP Code:

```
      <?php
      $array =
      array("name"=>"Toyota","type"=>"Celica","colour"=>"black","manufactured"=>
      "1991");
      $array2 = array("Toyota","Celica","black","1991");
      $array3 = array("name"=>"Toyota","Celica","colour"=>"black","1991");
      print_r($array); print_r($array2); print_r($array3);
      ?>
```

PHP Output: Array

```
(
[name] => Toyota [type] => Celica [colour] => black
[manufactured] => 1991
)
Array
(
[0] => Toyota [1] => Celica [2] => black [3] => 1991
)
Array
(
[name] => Toyota [0] => Celica [colour] => black [1] => 1991
)
```

Array ([name] => Toyota [type] => Celica [colour] => black [manufactured] => 1991) Array ([0] => Toyota [1] => Celica [2] => black [3] => 1991) Array ([name] => Toyota [0] => Celica [colour] => black [1] => 1991)

Example #2

The following example will output the identical text as Example #1:

```
      <?php
      $array['name']="Toyota";
      $array['type']="Celica";
      $array['colour']="black";
      $array['manufactured']="1991";
      $array2[]="Toyota";
      $array2[]="Celica";
      $array2[]="black";
      $array2[]="1991";
      $array3['name']="Toyota";
      $array3[]="Celica";
      $array3['colour']="black";
      $array3[]="1991";
      print_r($array); print_r($array2); print_r($array3);
      ?>
```

Example #3

Using the Example #1 and Example #2 above, now you can try and use arrays the same way as normal variables:

PHP Code:

```
<?php
echo "Manufacturer: &lt;b&gt;{$array['name']}&lt;/b&gt;&lt;br /&gt;\n";
echo "Brand: &lt;b&gt;{$array2['1']}&lt;/b&gt;&lt;br /&gt;\n";
echo "Colour: &lt;b&gt;".$array3['colour']."&lt;/b&gt;&lt;br /&gt;\n";
echo "Year Manufactured: &lt;b&gt;".$array3[1]."&lt;/b&gt;&lt;br /&gt;\n"
?>
```

PHP Output:

Manufacturer: <b>Toyota</b><br /> Brand: <b>Celica</b><br />
Colour: <b>black</b><br />
Year Manufactured: <b>1991</b><br />

HTML Render: Manufacturer: Toyota Brand: Celica
Colour: black
Year Manufactured: 1991

Multidimensional Arrays

Elements in an array can also be an array, allowing for multidimensional arrays. An example, in accordance with the motoring examples above, is:

```
<?php
$cars = array(
"car1" => array("make" => "Toyota","colour" => "Green","year" =>
1999,"engine_cc" =>
1998),
"car2" => array("make" => "BMW","colour" => "RED","year" =>
2005,"engine_cc" =>
2400),
"car3" => array("make" => "Renault","colour" => "White","year" =>
1993,"engine_cc" =>
1395),
);
?>
```

In this example, if you were to use:

```
<?php
echo "$cars['car1']['make']<br>";
echo "$cars['car3']['engine_cc']";
?>
```

The output would be: Toyota
1395

Array Functions

There exist dozens of array manipulation functions. Before implementing your own, make sure it doesn't already exist as a PHP function in Array functions (PHP manual entry).

Array traversal

In various circumstances, you will need to visit every array element and perform a task upon it.

The simplest and the most widely used method for this is the foreach operator which loops through the whole array and works individually with each key/item couple. If a more complex way of traversing the array is needed, the following functions operate using the internal array pointer:

reset - sets the internal pointer to the first element and returns the first element prev - sets the internal pointer to the previous element and returns it

current - returns the current element; does not change the internal pointer next - sets the internal pointer to the next element and returns it

each - returns the current element; then sets the internal pointer to the next element end - sets the internal pointer to the last element and returns the last element

```
<?php
// using an array's iterator to print its values in reverse order
$my_array = array('a', 'b', 'c');
end($my_array);
while($i = current($my_array)) { echo $i."\n"; prev($my_array);
}
?>
```

Another possibility is defining a function and applying it to each array element via one of the following functions:

array_walk - applies a function to each array element

array_walk_recursive - same, but if the element is itself an array, it will traverse that array too

Control structures

if structure
The if Statement

Conditional structures are used to control which statements get executed. They are composed of three fundamental elements:

if statements;

elseif statements; and else statements.

Conditionals in PHP are structured similarly to those found in C++ and Java. The structure begins with an if clause, which is composed of the word "if" followed by a true/false statement in parentheses ( ). The subsequent code will be contained in a block, denoted by curly braces { }. Sometimes the braces are omitted, and only one line will follow the if statement. elseif and else clauses sometimes occur after the if clause, to test for different statements.

The if clause says "If this statement is true, I want the program to execute the following statements. If it is false, then ignore these statements." In technical terms, it works like this: When an if statement is encountered, the true/false statement in parentheses is evaluated. If the statement is found to be true, the subsequent block of code contained in curly braces is executed. However, if the statement is found to be false, the program skips those lines and executes the next non-blank line.

Following the if clause are two optional clauses: else and elseif. The elseif clause says "If the last statement was false, let's see if this statement is true. If it is, execute the following code. If it isn't, then skip it." elseif statements are only evaluated when the preceding if statement comes out to be false. Otherwise they are skipped. Other than that, the elseif clause works just like a regular if clause. If it is true, its block is executed, if not, its block is skipped.

Finally, the else clause serves as a "catch-all" for an if statement. Essentially the else statement says "If all of the preceding tests fail, then execute this code."

Example 1

```
<?php
$foo = 1;
$bar = 2;
if($foo == $bar) {
echo "$foo is equal to $bar.";
} elseif ($foo > $bar) {
echo "$foo is greater than $bar.";
} else {
echo "$foo is less than $bar.";
}
?>
```

Example 2

```
<?php
$lower = 10;
$upper = 100;
$needle = 25;
if(($needle >= $lower) && ($needle <= $upper)) {
echo "The needle is in the haystack.";
} else if(($needle <= $lower) || ($needle >= $upper)) {
echo "The needle is outside of the haystack.";
}
?>
```

Conditional Expressions

Conditional Values function via basic formal logic. It is important to understand how the if clause, among other clauses, evaluates these conditional values.

It is easiest to examine such with boolean values in mind, meaning that the result of a conditional value will be either TRUE or FALSE and not both. For example, if variable $x = 4, and a conditional structure is called with the expression if($x == 4), then the result of the expression will be TRUE, and the if structure will execute. However, if the expression is ($x == 0), then the result will be FALSE, and the code will not execute. This is simple enough.

This becomes more complicated when complex expressions are considered. The two basic operators that expressions can be conjoined with are the AND (&&) and OR (||).

Examples

We are given variables $x and $y.

```
$x = 4;
$y = 8;
```

Given the complex expression: ($x == 4 AND $y == 8)

We are given a result of TRUE, because the result of both separate expressions are true. When expressions are joined with the AND operator, both sides MUST be true for the whole expression to be true.
Similarly:

($x == 4 OR $y == 8)

We are given a result of TRUE as well, because at least one expression is true. When expressions are joined with the OR operator, at least one side MUST be true for the whole expression to be true.
Conversely,
($x == 4 AND $y == 10)

This expression will return FALSE, because at least one expression in the whole is false. However,
($x == 4 OR $y == 10)
This expression will return TRUE, because at least one expression in the whole is true.

Code Blocks

A code block is one or more statements or commands that are contained between a pair of curly braces { }. Blocks are used primarily in loops, conditionals and functions. Blocks can be nested inside one another, for instance as an if structure inside of a loop inside of a function.

If, after one of the conditional statements, there is no block of code enclosed by curly braces, only the next statement will be executed. It is recommended that you avoid using this to help prevent accidents when adding extra code after the block.
The following code will not work as intended:

```
if(FALSE)
echo 'FALSE evaluates to true.';
echo 'Who knew that FALSE was TRUE?';
```

The second echo statement was executed, despite the if clause. The lack of brackets caused the if statement to only apply to the first statement, making the second statement evaluate regardless of the outcome of the if statement.

To avoid this problem, make sure to use brackets with conditional statements, even if there is only a single line of code to be executed. This prevents the error in the above code from occurring when you add an extra line after the existing block.
This code fixes the previous bug.

```
if(FALSE)
{
echo 'FALSE evaluates to true.';
echo 'Who knew that FALSE was TRUE?';
}
```

Switch Cases structure

Here's an example of a simple game where a user enters a $user_command and different functions are run as a result:

```
        if($user_command == "n")
        {
        go_north();
        }
        else if($user_command == "e")
        {
        go_east();
        }
        else if($user_command == "s")
        {
        go_south();
        }
        else if($user_command == "w")
        {
        go_west();
        }
        else
        {
        do_something_else();
        }
```

Clearly, there's a lot of repeated code here. The switch case structure allows you to avoid this redundant code. It allows programmers to repeatedly compare the value of a certain variable to a list of possible values and execute code based on the result. This is the syntax for a switch case statement, compared to the same code written using if statements:if statement style switch case style

```
if($firstvariable == 'comparison1'
|| $firstvariable == 'comparison2')
{
doSomething();
doSomethingElse();
}
else if ($firstvariable == 'comparison3')
{
doAThirdThing();
}
else
{
launchMissiles();
}          // Look at how much switch case saves you!
switch($firstvariable)
{
case 'comparison1': case 'comparison2': doSomething();
doSomethingElse();
break;
case 'comparison3': doAThirdThing(); break;
default: launchMissiles(); break;
}
```

The switch case style will save you from retyping $firstvariable, and make your code look cleaner (especially if that code is a long chain of simple if statements). Returning to our zork sample program, we have:Original Code Switch-Case Code

```
if($user_command == "n")
{
go_north();
}
else if($user_command == "e")
{
go_east();
}
else if($user_command == "s")
{
go_south();
}
else if($user_command == "w")
{
go_west();
}
else
{
do_something_else();
}          switch($user_command)
{
case 'n': go_north(); break;
case 'e': go_east(); break;
case 's': go_south(); break;
case 'w':
go_west();
break; default: do_something_else(); break;
}

Syntax switch($var)
{
case [value]: [code]
break;

case [value]: [code]
break;
... default:
[code]
break;
}
```

In this example. $var is the first variable to be compared. This variable is then compared against each case statement from the top down, until it finds a match. At that point, the code will excecute until a break statement is reached (which will allow you to leave the case statement entirely).

Important Warning about Using Switch Case Statements

Don't forget to use break when you mean break! If you forget, you might run functions you don't intend to. However, there are circumstances where leaving breaks out can be useful. Consider this example:

```
switch ($n) {
case 0: case 1: case 2:
//only executes if $n is 0, 1 or 2 doSomethingForNumbers2OrSmaller(); break;
case 3:
//only executes if $n is 3 doSomethingForNumber3(); default:
//only executes if $n is 3 or above doSomethingForNumbers3OrBigger(); break;
}
```

This kind of coding is sometimes frowned upon, since it's not always as clear to see what the code is meant to do. Also, consider commenting case statements that aren't supposed to have a break; statement before the next case, so when others look at your code, they know not to add a break.

While loop

```
<?php
$c = 0;
while ($c < 5) {
echo $c++;
} ?>

Example 2
<?php
$myName="Fred";
while ($myName!="Rumpelstiltskin") {
if ($myName=="Fred") {
$myName="Leslie";
}
else {
$myName="Rumpelstiltskin";
}
}
echo "How did you know?\n";
?>
```

Analysis Example 1

This is an example that prints the numbers from 0 to 4. $c starts out as 0. When the while loop is encountered, the expression $c < 5 is evaluated for truth. If it is true, then it executes what is in the curly braces. The echo statement will print 0, and then add one to $c. The program will then go back to the top of the loop and check the expression again. Since it is true again, it will then return 1 and add one to $c. It will keep doing this until $c is equal to 5, where the statement $c<5 is false. After that, it finishes.

Example 2

The first line of the program sets $myName to "Fred". After that, the while statement checks if $myName equals "Rumpelstiltskin". The != means 'does not equal', so the expression is true, and the while loop executes its code block. In the code block an if statement (see previous chapter) checks if it equals Fred. Since it does, it then reassigns

$myName to equal "Leslie". Then it skips the else, since the if was true and evaluated. Then it reaches the end of the loop, so it goes back and checks if $myName does not equal "Rumpelstiltskin". Since it still doesn't, it's true, and then it goes into the loop again. This time, the if statement is false, so the else is executed. This sets $myName to "Rumplestiltskin". We again get to the end of the loop, so it goes back, and checks. Since

$myName does equal "Rumpelstiltskin", the while condition is false, and it skips the loop and continues on, where it echos, "How did you know?"

Loops

Loops are another important basic programming technique. Loops allow programs to execute the same lines of code repeatedly, this is important for many things in programs. In PHP you often use them to layout tables in HTML and other similar functions.

Infinite Loops

Loops can be very handy, but also very dangerous! Infinite loops are loops that fail to exit, causing them to execute until the program is stopped. This is caused when no matter what occurs during the loop, the condition will never become false and therefore never exit. For example, if Example 1 subtracted 1 from $c...

```
$c=0;
while ($c<5) {
$c--;
echo $c;
}
```

$c will always be less than 5, no matter what, so the loop will continue forever. This causes problems for those of us who don't have infinite time (or computer memory!). So, in that case, let's learn a handy dandy little bugger.

If you add 'break;' to a loop, the loop will end, no matter whether the condition is false or not.

Let's combine the two examples. Before moving on take a few moments to write out the steps this program goes through and what its output is.

```
<?php
$c = 1;
$myName="Fred";
while ($myName!="Rumplestilskin") {
if ($myName=="Fred") {
$myName="Leslie";
}
else {
$myName="Marc";
}
$c++;
if ($c==3) {
break;
}
}
echo "You lose and I get your baby!\n";
?>
```

Do while loop.

The do while loop is similar in syntax and purpose to the while loop. The do/while loop construct moves the test that continues the loop to the end of the code block. The code is executed at least once, and then the condition is tested. For example:

```
<?php
$c = 6;
do {
echo 'Hi';
} while ($c < 5);
?>
```

Even though $c is greater than 5, the script will echo "Hi" to the page one time. PHP's do/while loop is not commonly used.

The continue statement

The continue statement causes the current iteration of the loop to end, and continues execution where the condition is checked - if this condition is true, it starts the loop again.

For loop

The for loop is one of the basic looping structures in most modern programming languages. Like the while loop, for loops execute a given code block until a certain condition is met.

The basic syntax of the for loop in PHP is similar to the c syntax:

```
      for([initialization]; [condition]; [step])
```

Initialization happens the first time the loop is run. It is used to initialize variables or perform other actions that are to be performed before the first execution of the body of the loop.

The Condition is evaluated before each execution of the body of the loop; if the condition is true, the body of the loop will be executed, if it is false, the loop is exited and program execution resumes at the first line after the body of the loop.

Step specifies an action that is to be performed after each execution of the loop body. Consider this

```
      for($i = 0; $i < 5; $i++)
      {
      echo($i . "<br />");
      }
```

PHP Output:

```
0<br />1<br />2<br />3<br />4<br />
```

HTML Render:

```
0
1
2
3
4
```

The loop can also be formatted without using concatenation, according to personal preference:

```
      for($i = 0; $i < 5; $i++)
      {
      echo "$i <br />";
      }
```

Explanation

The variable $i is initialized as 0. When the loop ran once for the first time, it printed the original value of $i, 0. Each time the loop ran, it incremented $i, then checked to see if $i was still less than 5. If it was, it continued looping. When $i finally reached 5, it was no longer less than 5, so PHP stopped looping and went to the next line after the loop code block (none, in this case).

Do note that the Initialization, Condition and Step of the For-loop can be empty. In this case, you will get an infinite loop, unless you use the "break" keyword inside the loop somwhere.

Also note that the Initialization and Step parts of the For-loop can hold more than one instruction. More info can be found via the link to the PHP manual at the end of this page.

Using FOR loops to traverse arrays

In the section on while loops the sort() example uses a while loop to print out the contents of the array. Generally programmers use for loops for this kind of job.

NOTE: Use of indices like below is highly discouraged. Use the key-value for-loop construct.

```
$menu = array("Toast and Jam", "Bacon and Eggs", "Homefries", "Skillet", "Milk and
Cereal");
// note to self: get breakfast after writing this article
      $count = count($menu);
      for($i = 0; $i < $count; $i++)
      {
      echo($i + 1 . ". " . $menu[$i] . "<br />");
      }

      Again, this can be formatted without concatenation if you prefer:
      for($i = 0; $i < $count; $i++)
      {
      $j=$i+1;
      echo "$j. {$menu[$i]} <br />";
      }
```

```
$count = count($menu);
```

We define the count before the for loop for more efficient processing. This is because each time the for loop is run (whilst $i < $count) it evaluates both sides of the equation and executes any functions. If we put $i < count($menu), This would evaluate count($menu) each time the process is executed which is costly when dealing with large arrays.

```
for($i = 0; $i < $count; $i++)
```

This line sets up the loop. It initializes the counter, $i, to 0 at the start, adds one every time the loop goes through, and checks that $i is less than the size of the array.

```
{
echo($i + 1 . ". " . $menu[$i] . "<br />");
}
```

The echo statement is pretty self-explanatory, except perhaps the bit at the start, where we echo $i + 1. We do that because, as you may recall, arrays start at 0 and end at n - 1 (where n is their length), so to get a numbered list starting at one, we have to add one to the counter each time we print it.

Of course, as I mentioned before, both pieces of code produce the following output:

1. Toast and Jam
2. Bacon and Eggs
3. Homefries
4. Skillet
5. Milk and Cereal

Foreach loop

1 The Code
2 Analysis
2.1 simple foreach statement
2.2 foreach with key values

```
foreach ($array as $someVar) {
echo ($someVar . "<br />");
}
or:
foreach ($array as $key => $someVar) {
echo ($key."holds the value ".$someVar."<br />");
}
```

The foreach loop is a special form of the standard for loop. The example above will print all the values of $array. The foreach structure is a convenient way to loop through an array.

simple foreach statement

Foreach loops are useful when dealing with an array indexed with arbitrary keys (e.g. non- numeric ones):

```
$array = array(
"1st" => "My House", "2nd" => "My Car", "3rd" => "My Lab"
);
```

To use the classical for structure, you'd have to write:

```
// get all the array keys
$arrayKeys = array_keys($array);
// loop through the keys
for ($i=0; $i<count($array); $i++) {
// get each array value using its key echo $array[($arrayKeys[$i])] . "<br />";
}
```

Basically, an array value can be accessed only from its key: to make sure you get all the values, you first have to make a list of all the existings keys then grab all the corresponding values. The access to first aray value, the previous example does the following steps:

```
$firstKey = $arrayKeys[0]; // which is '1st'
$firstValue = $array[$firstKey]; // which is 'My House' ($array('1st'))
The foreach structure does all the groundwork for you:
foreach ($array as $someVar) {
echo $someVar . "<br />";
}
```

Note how the latter example is easier to read (and write). Both will output: My House
My Car
My Lab

foreach with key values

If you need to use the array keys in your loop, just add the variable as in the following statement:

```
foreach ($array as $myKey => $value) {
// use $myKey
}
```

Note that this is very usefull when constructing a dropdown list in HTML. You can use a foreach-loop to have the $myKey variable inserted into the value="..." part and the $value as the actuall text. This form mimics the way we used custom keys for $array elements. It will not only assign the elements of $array to $someVar, but also assign the keys of those elements to $i.

```
<?php
$array = array("1st" => "My House", "2nd" => "My Car", "3rd" => "My Lab");
foreach ($array as $i => $someVar) {
echo $i . ": " . $someVar . "<br />\n";
}
?>
```

PHP Output:

```
1st: My House<br />
2nd: My Car<br />
3rd: My Lab<br />
```

HTML Render:

```
1st: My House
2nd: My Car
3rd: My Lab
```

Note that if you change the assigned variable inside the foreach loop, the change will not be reflected to the array. Therefore if you need to change elements of the array you need to change them by using the array key.

Example:

```
$array = array(
"1st" => "My House", "2nd" => "My Car",
"3rd" => "My Lab"
);

foreach ($array as $i => $someVar) {

// OK
if($someVar == 'My Lab')
$array[$i] = 'My Laboratory';

// doesn't update the array
$someVar = 'Foo';
}
```

Functions

Functions (often called methods) are a way to group common tasks or calculations to be re-used easily.

Functions in computer programming are much like mathematical functions: You can give the function values to work with and get a result without having to do any calculations yourself.

You can also find a huge list of predefined functions built into PHP in the PHP Manual's function reference.

Note that echo is not a function. "Calling a function" means causing a particular function to run at a particular point in the script. The basic ways to call a function include:

calling the function to write on a new line (such as after a ";" or "}")

print('I am Naresh, I am.');

calling the function to write on a new line inside a control

```
<?php
if ($a==72){
print('I am Naresh, I am.');
}
?>
```

assigning the returned value of a function to a variable "$var = function()"

```
<?php
$result = sum ($a, 5);
?>
```

calling a function inside the argument parentheses (expression) of a control

```
<?php
while ($i < count($one)){
}
?>
```

In our earlier examples we have called several functions. Most commonly we have called the function print() to print text to the output. The parameter for echo has been the string we wanted printed (for example print("Hello World!") prints "Hello World!" to the output).

If the function returns some information we assign it to a variable with a simple =:

$var1 = func_name(); [edit]

Parameters

Parameters are variables that exist only within that function. They are provided by the programmer when the function is called and the function can read and change them locally (except for reference type variables, which are changed globally - this is a more advanced topic).

When declaring or calling a function that has more than one parameter, you need to separate between different parameters with a comma ','.

A function declaration can look like this:

```
function print_two_strings($var1, $var2)
{
echo $var1; echo "\n"; echo $var2; return NULL;
}
```

To call this function you must give the parameters a value. It doesn't matter what the value is, as long as there is one.

function call: print_two_strings("hello", "world"); Output:

hello world

When declaring a function you sometimes want to have the freedom not to use all the parameters, therefore PHP allows you to give them default values when declaring the function:

```
function print_two_strings($var1 = "Hello World", $var2 = "I'm Learning PHP")
{
      echo($var1); echo("\n"); echo($var2);
}
```

These values will only be used if the function call does not include enough parameters. If there is only one parameter provided then $var2 = "I'm Learning PHP".

function call: print_two_strings("hello"); Output:

hello

I'm Learning PHP

Another way to have a dynamic number of parameters is to use PHP's built-in func_num_args, func_get_args, and func_get_arg functions.

```
function mean()
{
      $sum = 0;
      $param_count = func_num_args();
      for ($i = 0; $i < $param_count; $i++)
      {
      $sum += func_get_arg($i);
      }
      $mean = $sum / $param_count;
      echo "Mean: {$mean}";
      return NULL;
}
```

Or:

```
function mean()
{
      $sum = 0;
      $vars = func_get_args();
      for ($i = 0; $i < count($vars); $i++)
      {
      $sum += $vars[$i];
      }
      $mean = $sum / count($vars);
      echo "Mean: {$mean}";
      return NULL;
}
```

The above functions would calculate the arithmetic mean of all of the values passed to them and output it. The difference is that the first function uses func_num_args and func_get_arg, while the second uses func_get_args to load the parameters into an array. The output for both of them would be the same. For example:

```
mean(35, 43, 3);
```

Output: Mean: 27

==Returning a value==test

This function is all well and good, but usually you will want your function to return some information. Generally there are 2 reasons why a programmer would want information from a function:

The function does tasks such as calculations, and we need the result.

A function can return a value to indicate if the function encountered any errors. To return a value from a function use the return() statement in the function.

```
function add_numbers($var1 = 0, $var2 = 0, $var3 = 0)
{
$var4 = $var1 + $var2 + $var3;
return $var4;
}
```

Example PHP script:

```
<?php
function add_numbers($var1 = 0, $var2 = 0, $var3 = 0)
{
$var4 = $var1 + $var2 + $var3;
return $var4;
}

$sum = add_numbers(1,6,9);
echo "The result of 1 + 6 + 9 is {$sum}";
?>
```

Result:

```
The result of 1 + 6 + 9 is 16
```

Notice that a return() statement ends the function's course. If anything appears in a function declaration after the return() statement is executed, it is parsed but not executed. This can come in handy in some cases. For example:

```
<?php
function divide ($dividee, $divider) {
if ($divider == 0) {
//Can't divide by 0. return false;
}
$result = $dividee/$divider;
return $result;
}
?>
```

Notice that there is no else after the if. This is due to the fact that if $divider does equal 0, the return() statement is executed and the function stops.

If you want to return multiple variables you need to return an array rather than a single variable. For example:

```
<?php
function maths ($input1, $input2) {
$total = ($input1 + $input2);
$difference = ($input1 - $input2);
$ret = array("tot"=>$total, "diff"=>$difference);
return $ret;
}
?>
```

When calling this from your script you need to call it into an array. For example:

```
<?php
$return=maths(10, 5);
?>
```

In this case $return['tot'] will be the total (eg 15), while $return['diff'] will be the difference (5).

Runtime function usage

A developer can create functions inside a PHP script without having to use the function name($param...) {} syntax. This can done by way of programming that can let you run functions dynamically.

Executing a function that is based on a variable's name

There are two ways to do it, either using direct call or call_user_func or call_user_func_array:

Using call_user_func* functions to call functions

call_user_func and call_user_func_array only differ that the call_user_func_array allows you to use the second parameter as array to pass the data very easily, and call_user_func has an infinite number of parameters that is not very useful in a professional way. In these examples, a class will be used for a wider range of using the example:

```
<?php
class Some_Class {
function my_function($text1,$text2,$text3) {
$return = $text1."\n\n".$text2."\n\n".$text3;
return $return;
}
}
$my_class=new Some_Class();
?>
```

Using call_user_func:

```
<?php
$one = "One";
$two = "Two";
$three = "Three";
$callback_func = array(&$my_class,"my_function");
$result = call_user_func($callback_func,$one,$two,$three);
echo $result;
?>
```

Using call_user_func_array:

```
<?php
$one = "One";
$two = "Two";
$three = "Three";
$callback_func = array(&$my_class,"my_function");
$result = call_user_func_array($callback_func,array($one,$two,$three));
echo $result;
?>
```

Note how call_user_func and call_user_func_array are used in both of the examples. call_user_func_array allows the script to execute the function more dynamically.

As there was no example of using both of these functions for a non-class function, here they are:
Using call_user_func:

```
<?php
$one = "One";
$two = "Two";
$three = "Three";
$callback_func = "my_function";
$result = call_user_func($callback_func,$one,$two,$three);
echo $result;
?>
```

Using call_user_func_array:

```
<?php
$one = "One";
$two = "Two";
$three = "Three";
$callback_func = "my_function";
$result = call_user_func_array($callback_func,array($one,$two,$three));
echo $result;
?>
```

More complicated examples

```
<?php
$my_func($param1, $param2);
$my_class_name = new ClassObject();
$my_class_name->$my_func_from_that_class($param1, $param2);
// The -> symbol is a minus sign follow by a "larger then" sign. It allows you to use a function that is defined in a different PHP class.
// It comes directly from Object-Oriented programming. Via a constructor, a function of that class is executable.
// This specific example is a function that returns no values. call_user_func($my_func, $param1, $param2);

call_user_func(array(&${$my_class_name}, $my_func), $param1, $param2);

call_user_func_array($my_func, array($param1, $param2));
// Most powerful, dynamic example
call_user_func_array(array(&${$my_class_name}, $my_func), array($param1, $param2));
?>
<?php
function positif ($x + $y;) {
$x=2;
$y=5;
$z = $x+$y;
echo $z;
}
positif=$x+$y;
?>
```

TODO: Somebody add what the & in codeline 5 means

Creating runtime functions

Creating runtime functions is a very good way of making the script more dynamic:

```
<?php
$function_name=create_function('$one, $two','return $one+$two;');
echo $function_name."\n\n";
echo $function_name("1.5", "2");
?>
```

create_function creates a function with parameters $one and $two, with a code to evaluate return... When create_function is executed, it stores the function's info in the memory and returns the function's name. This means that you cannot customise the name of the function although that would be preferred by most developers.

Files

Working with files is an important part of any programming language and PHP is no different. Whatever your reasons are for wanting to manipulate files, PHP will happily accommodate them through the use of a handful of functions. You should have read and be comfortable with the concepts presented in the first five sections of this book before beginning here.

Fopen() and Fclose()

Fopen() is the basis for file manipulation. It opens a file in a certain mode (which you specify) and returns a handle. Using this handle you can read and/or write to the file, before closing it with the fclose() function.

Example Usage

```
<?php
$Handle = fopen('data.txt', 'r'); // Open the file for reading fclose($Handle); // Close the file
?>
```

In the example above you can see the file is opened for reading by specifying 'r' as the mode. For a full list of all the modes available to fopen() you can look at the PHP Manual page.

Opening and closing the file is all well and good but to perform useful operations you need to know about fread() and fwrite().

When a PHP script finishes executing, all open files are automatically closed. So although it is not strictly necessary to close a file after opening it, it is considered good programming practice to do so.

Reading

Reading can be done a number of ways. If you just want all the contents of a file available to work with you can use the file_get_contents() function. If you want each line of the file in an array you can use the file() command. For total control over reading from files fread() can be used.

These functions are usually interchangeable and each can be used to perform each other's function. The first two do not require that you first open the file with fopen() or then close it with fclose(). These are good for quick, one-time file operations. If you plan on performing multiple operations on a file it is best to use fopen() in conjunction with fread(), fwrite() and fclose() as it is more efficient.
An example of using file_get_contents() Code:

```
<?php
$Contents = file_get_contents('data.txt');
echo $Contents;
?>
```

Output:

```
I am the contents of data.txt
```

This function reads the entire file into a string and from then on you can manipulate it as you would any string.

An example of using file() Code:

```
<?php
$Lines = file('data.txt');
foreach($Lines as $Key => $Line) {
$LineNum = $Key + 1;
echo "Line $LineNum: $Line";
}
?>
```

Output:

```
Line 1: I am the first line of file
Line 2: I am the second line the of the file
Line 3: If I said I was the fourth line of the file, I'd be lying
```

This function reads the entire file into an array. Each item in the array corresponds to a line in the file.

An example of using fread() Code:

```
<?php
$Handle = fopen('data.txt', 'r');
$String = fread($Handle, 64);
fclose($Handle);

echo $String;
?>
```

Output:

```
I am the first 64 bytes of data.txt (if it was ASCII encoded). I
```

This function can read up to the specified number of bytes from the file and return it as a string. For the most part, the first two functions will be preferable but there are occasions when this function is needed.

As you can see, with these three functions you are able to easily read data from a file into a form that is convenient to work with. The next part shows how these functions can be used to do the jobs of the others but this is optional. You may skip it and move onto the
Writing section if you are not interested.

.

```
<?php
$File = 'data.txt';

function DetectLineEndings($Contents) {
if(false !== strpos($Contents, "\r\n")) return "\r\n"; elseif(false !== strpos($Contents, "\r")) return "\r";
else return "\n";
}

/* This is equivalent to file_get_contents($File) but is less efficient */
$Handle = fopen($File, 'r');
$Contents = fread($Handle, filesize($File));
fclose($Handle);
```

```
/* This is equivalent to file($File) but requires you to check for the line-ending type. Windows systems use \r\n, Macintosh \r and Unix \n. File($File) will automatically detect line-endings whereas fread/file_get_contents won't */
$LineEnding = DetectLineEndings($Contents);
$Contents = file_get_contents($File);
$Lines = explode($LineEnding, $Contents);

/* This is also equivalent to file_get_contents($File) */
$Lines = file($File);
$Contents = implode("\n", $Lines);

/* This is equivalent to fread($File, 64) if the file is ASCII encoded */
$Contents = file_get_contents($File);
$String = substr($Contents, 0, 64);
?>
```

Writing

Writing to a file is done by using the fwrite() function in conjunction with fopen() and fclose(). As you can see, there aren't as many options for writing to a file as there are for reading from one. However, PHP 5 introduces the function file_put_contents() which simplifies the writing process somewhat. This function will be discussed later in the PHP 5 section as it is fairly self-explanatory and does not require discussion here.

The extra options for writing don't come from the amount of functions, but from the modes available for opening the file. There are three different modes you can supply to the fopen() function if you wish to write to a file. One mode, 'w', wipes the entire contents of the file so anything you then write to the file will fully replace what was there before. The second mode, 'a' appends stuff to the file so anything you write to the file will appear just after the original contents of the file. The final mode 'x' only works for non-existent files. All three writing modes will attempt to create the file if it doesn't exist whereas the 'r' mode will not.

An example of using the 'w' mode

Code:

```
<?php
$Handle = fopen('data.txt', 'w'); // Open the file and delete its contents
$Data = "I am new content\nspread across\nseveral lines.";
fwrite($Handle, $Data);
fclose($Handle);

echo file_get_contents('data.txt');
?>
```

Output:

```
I am new content spread across several lines.
```

An example of using the 'a' mode

Code:

```
<?php
$Handle = fopen('data.txt', 'a'); // Open the file for appending
$Data = "\n\nI am new content."; fwrite($Handle, $Data); fclose($Handle);
echo file_get_contents('data.txt');
?>
```

Output:

```
I am the original content. I am new content.
```

An example of using the 'x' mode

Code:

```
<?php
$Handle = fopen('newfile.txt', 'x'); // Open the file only if it doesn't exist
$Data = "I am this file's first ever content!";
fwrite($Handle, $Data);
fclose($Handle);
echo file_get_contents('newfile.txt');
?>
```

Output:

```
I am this file's first ever content!
```

Of the three modes shown above, 'w' and 'a' are used the most but the writing process is essentially the same for all the modes.

Reading and Writing

If you want to use fopen() to open a file for both reading and writing all you need to do is put a '+' on the end of the mode. For example, reading from a file requires the 'r' mode. If you want to read and write to/from that file you need to use 'r+' as a mode. Similarly you can read and write to/from a file using the 'w+' mode however this will also truncate the file to zero length. For a better description visit the fopen() page which has a very useful table describing all the modes available.

Error Checking

Error checking is important for any sort of programming but when working with files in PHP it is especially important. This need for error checking arises mainly from the filesystem the files are on. The majority of webservers today are Unix-based and so, if you are using PHP to develop web-applications, you have to account for file permissions. In some cases PHP may not have permission to read the file and so if you've written code to read a particular file, it will result in an ugly error. More likely is that PHP doesn't have permission to write to a file and that will again result in ugly errors. Also, the file's existence is (somewhat obviously) important. When attempting to read a file, you must make sure the file exists first. On the other side of that, if you're attempting to create and then write to a file using the 'x' mode then you must make sure the file doesn't exist first.

In short, when writing code to work with files, always assume the worst. Assume the file doesn't exist and you don't have permission to read/write to it. In most cases this means you have to tell the user that, in order for the script to work, he/she needs to adjust those file permissions so that PHP can create files and read/write to them but it also means that your script can adjust and perform an alternative operation.

There are two main ways of error checking. The first is by using the '@' operator to suppress any errors when working with the file and then checking if the result is false or not. The second method involves using more functions like file_exists(), is_readable() and is_writeable().

Examples of using the '@' operator

```
<?php
$Handle = @ fopen('data.txt', 'r');
if(!$Handle) {
echo 'PHP does not have permission to read this file or the file in question doesn\'t exist.';
} else {
$String = fread($Handle, 64);
fclose($Handle);
}

$Handle = @ fopen('data.txt', 'w'); // The same applies for 'a' if(!$Handle) {
echo 'PHP either does not have permission to write to this file or it does not have permission to create this file in the current directory.';
} else {
fwrite($Handle, 'I can has content?');
fclose($Handle);
}

$Handle = @ fopen('data.txt', 'x');
if(!$Handle) {
echo 'Either this file exists or PHP does not have permission to create this file in the current directory.';
} else {
fwrite($Handle, 'I can has content?');
fclose($Handle);
}
?>
```

As you can see, the '@' operator is used mainly when working with the fopen() function. It can be used in other cases but is generally less efficient.

Examples of using specific checking functions

```
<?php
$File = 'data.txt';

if(!file_exists($File)) {
// No point in reading since there is no content
$Contents = '';

// But might want to create the file instead
$Handle = @ fopen($File, 'x'); // Still need to error-check ;)
if(!$Handle) {
echo 'PHP does not have permission to create a file in the current directory.';
} else {
fwrite($Handle, 'Default data');
fclose($Handle);
}
} else {
// The file does exist so we can try to read its contents if(is_readable($File)) {
$Contents = file_get_contents($File);
} else {
echo 'PHP does not have permission to read that file.';
}
}
```

```
if(file_exists($File) && is_writeable($File)) {
$Handle = fopen($File, 'w'); fwrite($Handle, 'I can has content?'); fclose($Handle);
}
?>
```

You can see by that last example that error-checking makes your code very robust. It allows it to be prepared for most situations and behave accordingly which is an essential aspect of any program or script.

Line-endings

Line-endings were mentioned briefly in the final example in the 'Reading' section of this chapter and it is important to be aware of them when working with files. When reading from a text file it is important to know what types of line-endings that file contains. 'Line- endings' are special characters that try to tell a program to display a new line. For example, Notepad will only move a piece of text to a new line if it finds "\r\n" just before the new line (it will also display new lines if you put word wrap on).

If someone writes a text file on a Windows system, the chances are that each line will end with "\r\n". Similarly, if they write the file on a Classic Macintosh (Mac OS 9 and under) system, each line will probably end with "\r". Finally, if they write the file on a Unix-based system (Mac OS X and GNU/Linux), each line will probably end with "\n".

Why is this important? Well, when you read a file into a string with file_get_contents(), the string will be one long line with those line-endings all over the place. Sometimes they will get in the way of things you want to do with the string so you can remove them with:

```
<?php
$String = str_replace(array("\n", "\r"), '', $String);
?>
```

Other times you may need to know what kind of line-ending is being used throughout the text in order to be consistent with any new text you add. Luckily, in 99% of cases, the line- endings will never change type throughout the text so the custom function 'DetectLineEndings' can be used as a quick way of checking:

```
<?php
function DetectLineEndings($String) {
if(false !== strpos($String, "\r\n")) return "\r\n"; elseif(false !== strpos($String, "\r")) return "\r"; else
return "\n";
}
?>
```

Most of the time though, it is just sufficient to be aware of their existence within the text so you can adjust your script to cope properly.

Binary Safe

So far, all of the text seen in this chapter has been assumed to be encoded in some form of plaintext encoding such as UTF-8 or ASCII. Files do not have to be in this format however and in fact there exist a huge number of formats that are aren't (such as pictures or executables). If you want to work with these files you have to ensure that the functions you are using are 'binary safe'.

Previously you would have to add 'b' to the end of the modes you used to tell PHP to treat the file as a binary file. Failing to do so would give unexpected results and generally 'weird-looking' data.

Since about PHP 4.3, this is no longer necessary as PHP will automatically detect if it needs to open the file as a text file or a binary file and so you can still follow most of the examples shown here. Working with binary data is a lot different to working with plaintext strings and characters and involves many more functions which are beyond the scope of this chapter however it is important you know about these differences.

Serialization

Serialization is a technique used by programmers to preserve their working data in a format that can later be restored to its previous form. In simple cases this means converting a normal variable such as an array into a string and then storing it somewhere. That data can then be unserialized and the programmer will be able to work with the array once again.

There is a whole chapter devoted to Serialization in this book as it is a useful technique to know how to use effectively. It is mentioned here as one of the primary uses of serialization is to store data on plain files when a database is not available. It is also used to store the state of a script and to cache data for quicker access later and files are one of the preferred media for this storage.

In PHP, serialization is very easy to perform through use of the serialize() and unserialize() functions. Here follows an example of serialization used in conjunction with file functions. An example of storing user details in a file so that they can be easily retrieved later.

Code:

```
<?php
/* This part of the script saves the data to a file */
$Data = array(
'id' => 114,
'first name' => 'Foo',
'last name' => 'Bartholomew',
'age' => 21,
'country' => 'England'
);
$String = serialize($Data);

$Handle = fopen('data.dat', 'w'); fwrite($Handle, $String); fclose($Handle);

/* Then, later on, we retrieve the data from the file and output it */
$String = file_get_contents('data.dat');
$Data = unserialize($String);

$Output = '';
foreach($Data as $Key => $Datum) {
$Field = ucwords($Key);
$Output .= "$Field: $Datum\n";
}
echo $Output
?> Output:
Id: 114
```

```
First Name: Foo
Last Name: Bartholomew
Age: 21
Country: England
```

PHP 5

There is one particular function specific to files that was introduced in PHP 5. That was the file_put_contents() function. It offers an alternative method of writing to files that does not exist in PHP 4. To see how it differs, it is easiest to just look at an example.

Examples showing writing to a file in PHP 4 and the equivalent in PHP 5 with the file_put_contents() function

```
<?php
$File = 'data.txt';
$Content = 'New content.';

// PHP 4, overwrite entire file with data
$Handle = fopen($File, 'w'); fwrite($Handle, $Content); fclose($Handle);

// PHP 5
file_put_contents($File, $Content);

// PHP 4, append to a file
$Handle = fopen($File, 'a'); fwrite($Handle, $Content); fclose($Handle);

// PHP 5
file_put_contents($File, $Content, FILE_APPEND);
?>
```

File_put_contents() will also attempt to create the file if it doesn't exist and it is binary safe. There is no equivalent of the 'x' mode for file_get_contents().

File_put_contents() is almost always preferable over the fopen() method except when performing multiple operations on the same file. It is more preferable to use it for writing than file_get_contents() is for reading and for this reason, a function is provided here to emulate the behaviour of file_put_contents() for PHP 4:

```
<?php if(!function_exists('file_put_contents')) {
function file_put_contents($File, $Data, $Append = false) {
if(!$Append) $Mode = 'w';
else $Mode = 'a';

$Handle = @ fopen($File, $Mode);
if(!$Handle) return false;

$Bytes = fwrite($Handle, $Data);
fclose($Handle);
return $Bytes;
}
}
?>
```

Mail

The mail function is used to send E-mail Messages through the SMTP server specified in the php.ini Configuration file.

bool mail (string to, string subject, string message [, string additional_headers [, string additional_parameters]])

The returned boolean will show whether the E-mail has been sent successfully or not. This example will send message "message" with the subject "subject" to email address "example@domain.tld". Also, the receiver will see that the eMail was sent from "Example2

<example2@domain.tld>" and the receiver should reply to "Example3
<example3@domain.tld>"

```
<?php mail(
"example@domain.tld", // E-Mail address
"subject", // Subject
"message", // Message
"From: Example2 <example2@domain.tld>\r\nReply-to: Example3
<example3@domain.tld>) // Additional Headers
;
?>
```

There is no requirement to write E-mail addresses in format "Name <email>", you can just write "email".

This will send the same message as the first example but includes From: and Reply-To: headers in the message. This is required if you want the person you sent the E-mail to be able to reply to you. Also, some E-mail providers will assume mail is spam if certain headers are missing so unless you include the correct headers your mail will end up in the junk mail folder.

Important notes

PHP by default does not have any mail sending ability itself. It needs to pass the mail to a local mail transfer agent, such as sendmail. This means you cannot just run PHP by itself and expect it to send mail; you must have a mail transfer agent installed.

Make sure you do not have any newline characters in the to or subject, or the mail may not be sent properly.

However, the additional headers field -- which should always include the From: header -- may also include other headers. On PHP for Windows, each header should be followed by

\r\n but on Unix versions, you should only include \r between header lines. Don't put \n or

\r\n after the final additional header line.

The to parameter should not be an address in the form of "Name

<someone@example.com>". The mail command may not parse this properly while talking with the MTA (Particularly under Windows).

Error Detection

Especially when sending multiple emails, such as for a newsletter script, error detection is important.

Use this script to send mail while warning for errors:

```
$result=@mail($to,$subject,$message,$headers);
if ( $result ) echo "Email sent successfully."
else echo "Email was not sent, as an error occurred."
```

Sending To Multiple Addresses Using Arrays

In the case below the script has already got a list of emails, we simply use the same procedure for using a loop in PHP with mysql results. The script below will attempt to send an email to every single email address in the array until it runs out.

```
while ($row = mysql_fetch_assoc($result)) {
mail($row['email'], $subject, $message, null,"-f$fromaddr");
}
```

Then if we integrate the error checking into the multiple email script we get the following

```
$errors = 0
$sent = 0

while ($row = mysql_fetch_assoc($result)) {
$result = "";
$result = @mail($row['email'], $subject, $message, null,"-f$fromaddr");
if ( !$result ) $errors = $errors + 1;
$sent = $sent + 1;
}
echo "You have sent $sent messages";
echo "However there were $errors messages";
```

Cookies

Cookies are small pieces of data stored as text on the client's computer. Normally cookies are used only to store small amounts of data. Even though cookies are not harmful some people do not permit cookies due to concerns about their privacy. In this case you have to use Sessions.

This lesson covers setting and retrieving data from cookies.

Setting a cookie

Setting a cookie is extremely easy with setcookie().

```
setcookie("test", "PHP-Hypertext-Preprocessor", time()+60, "/location",1);
```

Here the setcookie function is being called with four arguments (setcookie has 1 more optional argument, not used here). In the above code, the first argument is the cookie name, the second argument is the cookie contents and the third argument is the time after which the cookie should expire in seconds (time() returns current time in seconds, there time()+60 is one minute from now). The path, or location, element may be omitted, but it does allow you to easily set cookies for all pages within a directory, although using this is not generally recommended.

You should note that since cookies are sent with the HTTP headers the code has to be at the top of the page (Yes, even above the DOCTYPE declaration). Any other place will generate an error.

Retrieving cookie data

If a server has set a cookie the browser sends it to the server each time a page loads. The name of each cookie sent by your server is stored in the superglobal array _COOKIE. So in the above example the cookie would be retrieved by calling $_COOKIE['test']. To access data in the cookie we use explode(). explode() turns a string into an array with a certain delimiter present in the string. That is why we used those dashes(- hyphens) in the cookie contents. So to retrieve and print out the full form of PHP from the cookie we use the code:

```
<?php
$array = explode("-", $_COOKIE['test']); //retrieve contents of cookie print("PHP stands for
".$array[0].$array[1].$array[2]); //display the content
?>
```

Note: $_COOKIE was Introduced in 4.1.0. In earlier versions, use
$HTTP_COOKIE_VARS.

Where are cookies used?

Cookies can be often used for:
user preferences
inventories
quiz or poll results shopping carts
user authentication
remembering data over a longer period

You should never store unencrypted passwords in cookies as cookies can be easily read by the users.

You should never store critical data in cookies as cookies can be easily removed or modified by users.

Sessions

Sessions allow the PHP script to store data on the web server that can be later used, even between requests to different php pages. Every session has got a different identifier, which is sent to the client's browser as a cookie or as a $_GET variable. Sessions end when the user closes the browser, or when the web server deletes the session information, or when the programmer explicitly destroys the session. In PHP it's usually called PHPSESSID. Sessions are very useful to protect the data that the user wouldn't be able to read or write, especially when the PHP developer doesn't want to give out information in the cookies as they are easily readable. Sessions can be controlled by the $_SESSION superglobal. Data stored in this array is persistent throughout the session. It is a simple array. Sessions are much easier to use than cookies, which helps PHP developers a lot. Mostly, sessions are used for user logins, shopping carts and other additions needed to keep browsing smooth. PHP script can easily control the session's cookie which is being sent and control the

whole session data. Sessions are always stored in a unique filename, either in a temporary folder or in a specific folder, if a script instructs to do so.

Using Sessions

At the top of each php script that will be part of the current session there must be the function session_start(). It must be before the first output (echo or others) or it will result in an error "Headers already sent out".

```
session_start();
```

This function will do these actions:

It will check the _COOKIE or _GET data, if it is given

If the session file doesn't exist in the session.save_path location, it will : Generate a new Unique Identifier, and

Create a new file based on that Identifier, and

Send a cookie to the client's browser

If it does exist, the PHP script will attempt to store the file's data into _SESSION variable for further use

Now, you can simply set variables in 2 different ways, the default method:

$_SESSION['example'] = "Test";

Or the deprecated method:

$example="Test";

session_register($example);

Both of the above statements will register the session variable $_SESSION['example'] as "Test". The deprecated method should not be used, it is only listed because you can still see it in scripts written by programmers that don't know the new one. The default method is preferred.

Session Configuration Options

PHP Sessions are easy to control and can be made even more secure or less secure with small factors. Here are runtime options that can be easily changed using php_ini()

Function:Name	**Default**	**Changeable**
session.save_path "/tmp" PHP_INI_ALL		
session.name	"PHPSESSID"	PHP_INI_ALL
session.save_handler session.auto_start "0" session.gc_probability	"files" PHP_INI_ALL PHP_INI_ALL	
"1"	PHP_INI_ALL	
session.gc_divisor "100" PHP_INI_ALL		
session.gc_maxlifetime	"1440"PHP_INI_ALL	
session.serialize_handler "php" PHP_INI_ALL		
session.cookie_lifetime session.cookie_path session.cookie_domain session.cookie_secure session.use_cookies	"0" "/" "" "" "1"	PHP_INI_ALL PHP_INI_ALL PHP_INI_ALL PHP_INI_ALL PHP_INI_ALL PHP_INI_ALL PHP_INI_ALL
session.use_only_cookies "0"		
session.referer_check session.entropy_file"" session.entropy_length session.cache_limiter session.cache_expire session.use_trans_sid session.bug_compat_42	""	
PHP_INI_ALL		

"0"	PHP_INI_ALL	
"nocache"	PHP_INI_ALL	
"180" PHP_INI_ALL		
"0" "1"	PHP_INI_SYSTEM/PHP_INI_PERDIR PHP_INI_ALL	
session.bug_compat_warn	"1"	PHP_INI_ALL
session.hash_function	"0"	PHP_INI_ALL
session.hash_bits_per_character"4"	PHP_INI_ALL	
url_rewriter.tags	"a=href,area=href,frame=src,input=src,form=fakeentry"	

PHP_INI_ALL

A simple example of this use would be this code:

```
//Setting The Session Saving path to "sessions", must be protected from reading
session_save_path("sessions"); // This function is an alternative to
ini_set("session.save_path","sessions");
//Session Cookie's Lifetime ( not effective, but use! )
ini_set("session.cookie_lifetime",time()+60*60*24*500);
//Change the Session Name from PHPSESSID to SessionID
session_name("SessionID");
//Start The session
session_start();
//Set a session cookie ( Required for some browsers, as settings that had been done
before are not very effective
setcookie(session_name(), session_id(), time()+3600*24*365, "/");
```

This example simply sets the cookie for the next year

Ending a Session

When user clicks "Logout", or "Sign Off", you would usually want to destroy all the login data so nobody could have have access to it anymore. The Session File will be simply deleted as well as the cookie to be unset by:

```
        session_destroy();
```

Using Session Data of Other Types

Simple data such as integers, strings, and arrays can easily be stored in the $_SESSION superglobal array and be passed from page to page. But problems occur when trying to store the state of an object by assignment. Object state can be stored in a session by using the serialize() function. serialize() will write the objects data into an array which then can be stored in a $_SESSION supergloblal. unserialize() can be used to restore the state of an object before trying to access the object in a page that is part of the current session. If objects are to be used across multiple page accesses during a session, the object definition must be defined before calling unserialize(). Other issues may arise when serializing and unserializing objects.

Avoiding Session Fixation

Wikipedia has related information at

Session fixation

Session fixation describes an attack vector in which a malicious third-party sets (i.e. fixes) the session identifier (SID) of a user, and is thus able to access that user's session. In the base-level implementation of sessions, as described above, this is a very real vulnerability, and every PHP program that uses sessions for anything at all sensitive should take steps to remedy it. The following, in order of how widely applicable they are, are the measures to take to prevent session fixation:

Do not use GET or POST variables to store the session ID (under most PHP configurations, a cookie is used to store the SID, and so the programmer doesn't need to do anything to implement this);
Regenerate the SID on each user request (using session_regenerate_id() at the beginning of the session);
Use session time-outs: for each user request, store the current timestamp, and on the next request check that the timeout interval has not passed;
Provide a logout function;

Check the 'browser fingerprint' on each request. This is a hash, stored in a $_SESSION variable, comprising some combination of the user-agent header, client IP address, a salt value, and/or other information. See below for more discussion of the details of this; it is thought by some to be nothing more than 'security through obscurity'. [TODO]
Check referrer: this does not work for all systems, but if you know that users of your site must be coming from some known domain you can discard sessions tied to users from elsewhere. Relies on the user agent providing the Referrer header, which should not be assumed.

This example code addresses all of the above points save the referrer check.

```
$timeout = 3 * 60; // 3 minutes
$fingerprint = md5('SECRET-SALT'.$_SERVER['HTTP_USER_AGENT']);
session_start();
if ( (isset($_SESSION['last_active']) && (time() > ($_SESSION['last_active']+$timeout)))
|| (isset($_SESSION['fingerprint']) && $_SESSION['fingerprint']!=$fingerprint)
|| isset($_GET['logout']) ) {
do_logout();
}
session_regenerate_id();
$_SESSION['last_active'] = time();
$_SESSION['fingerprint'] = $fingerprint;
```

The do_logout() function destroys the session data and unsets the session cookie.

DATA BASES MySQL

MySQL is the most popular database used with PHP. PHP with MySQL is a powerful combination showing the real power of Server-Side scripting. PHP has a wide range of MySQL functions available with the help of a separate module. In PHP5, this module has been removed and must be downloaded separately.

MySQL allows users to create tables, where data can be stored much more efficiently than the way data is stored in arrays.

In order to use MySQL or databases in general effectively, you need to understand SQL, or Structured Query Language.

Note that this page uses the mysqli functions and not the old mysql functions. How to - Step By Step
Connecting to the MySQL server

PHP has the function mysqli_connect to connect to a MySQL server which handles all of the low level socket handling. We will supply 4 arguments; the first is the name of your MySQL server, the

second a MySQL username, third a MySQL password and last a database name. In this example, it is assumed your server is localhost. If you are running a web server on one system, and MySQL on another system, you can replace localhost with the IP address or domain name of the system which MySQL resides on (ensure all firewalls are configured to open the appropriate ports).
mysqli_connect returns a

link_identifier that we can now use for communicating with the database. We will store this link in a variable called $link.

```
<?php
$cxn = mysqli_connect ("localhost", "your_user_name", "your_password", "database_name");
?>
```

Running a Query

We have connected to the mysql server and then selected the database we want to use, now we can run an SQL query over the database to select information, do an insert,

update or delete. To do this we use mysqli_query. This takes two arguments: the first is our link_identifier and the second is an SQL query string. If we are doing a select sql

statement mysqli_query generates a resource or the Boolean false to say our query failed, and if we are doing a delete, insert or update it generates a Boolean, true or false, to say if that was successful or not.

The basic code for running a query is the php function "mysqli_query($cxn, $query)". The "$query" argument is a MySQL query. The database argument is a database connection(here, the connection represented by $cxn). For example, to return the query "SELECT * FROM customers ORDER BY customer_id ASC", you could write

```
<?php
mysqli_query($cxn, "SELECT * FROM customers ORDER BY customer_id ASC");
?>
```

However, this straightforward method will quickly become ungainly due to the length of MySQL queries and the common need to repeat the query when handling the return. All (or almost all) queries are therefore made in two steps. First, the query is assigned a variable (conventionally, this variable is named "$query" or "$sql_query" for purposes of uniformity and easy recognition), which allows the program to call simply "mysqli_query($cxn, $sql_query)".

```
$sql_query = "SELECT * FROM customers ORDER BY customer_id ASC";
```

Secondly, to handle the information returned from the query, practical considerations require that the information returned also be assigned to a variable. Again by convention rather than necessity (i.e. you could name it anything you wanted), this information is often assigned to "$result", and the function is called by the assignment of the variable.

It is important to understand that this code calls the function mysqli_query, in addition to assigning the return to a variable "$result". [NOTE: The queries that ask for information -- SELECT, SHOW, DESCRIBE, and EXPLAIN -- return what is called a resource. Other types of queries, which manipulate the database, return TRUE if the operation is successful and FALSE if not, or if the user does not have permission to access the table referenced.]

To catch an error, for debugging purposes, we can write:

```
<?php
$result = mysqli_query ($cxn, $sql_query);
or die (mysqli_error () . " The query was:" . $sql_query);
?>
```

If the function mysqli_query returns false, PHP will terminate the script and print an error report from MySQL (such as "you have an error in your SQL syntax") and the query.

Thus, our final code would be, assuming that there were a database connection named $cxn:

```
<?php
$sql_query = "SELECT * FROM customers ORDER BY customer_id ASC";
$result = mysqli_query ($cxn, $sql_query);
or die (mysqli_error () . " The query was:" . $sql_query);
?>
```

Putting it all together

In the previous sections we looked at 3 commands, but not at how to use them in conjunction with each other. So let's take a look at selecting information for a table in our mysql database called MyTable, which is stored in a mysql database called MyDB.

```
<?php

//Connect to the mysql server and get back our link_identifier
$link = mysql_connect ("your_database_host", "your_user_name",
"your_password");

//Now we select which database we would like to use mysql_select_db ("MyDB", $link);

//Our SQL Query
$sql_query = "Select * From MyTable";

//Run our sql query
$result = mysql_query ($sql_query, $link);

//Close Database Connection mysql_close ($link);

?>
```

Getting Select Query Information

Well that doesn't help, because what are we to do with $result? Well when we do a select query we select out information from a database we get back what is known as a resource, and that is what is stored in $result, our resource identifier. A resource is a special type of PHP variable, but lets look at how to access information in this resource.

We can use a function called mysql_fetch_assoc it takes one parameter, our resource identifier $result, and generates an associative array corresponding to the fetched row. Each column in the table corresponds to an index with the same name. We can now extract out information and print it like so:

```
<?php
//Connect to the mysql server and get back our link_identifier
$link = mysql_connect("localhost", "your_user_name", "your_password")
or die('Could not connect: ' . mysql_error());
```

```
//Now we select which database we would like to use mysql_select_db("MyDB") or die('could not
select database');

//Our SQL Query
$sql_query = "Select * From MyTable";

//Run our sql query
$result = mysql_query($sql_query)or die('query failed'. mysql_error());

//iterate through result
while($row = mysql_fetch_assoc($result))
{
//Prints out information of that row print_r($row);
echo $row['foo'];
//Prints only the column foo.
}
// Free resultset (optional)
mysql_free_result($result);

//Close the MySQL Link mysql_close($link);
?>
```

PHP + MySQL + Sphinx

Once you understand the basics of how MySQL functions with PHP you might want to start learning about full text search engines. Once your site gets large (millions of database records) MySQL queries will start to get painfully slow, especially if you use them to search for text with wildcards (ex:
WHERE content='%text%')

. There are many free/paid solutions to stop this problem.

A good open source full text search engine is Sphinx Search. There is a WikiBook on how to use it with PHP and MySQL that explains the concepts of how Indexing works. You might want to read it before reading the official documentation.

PostgreSQL

PostgreSQL is another popular database used with PHP.

The basic syntax of PostgreSQL is the same as that of MySQL, although some functions have been renamed:
mysqli_connect becomes pg_connect
mysql_select_db is deprecated; it is specified in the pg_connect syntax mysqli_query becomes pg_query
mysql_error becomes pg_last_error mysql_close becomes pg_close mysql_fetch_assoc becomes pg_fetch_assoc mysql_free_result becomes pg_free_result

PHP Data Objects

PHP Data Objects, also known as PDO, is an interface for accessing databases in PHP without tying code to a specific database. Rather than directly calling mysql_, mysqli_, and pg_ functions, developers can use the PDO interface, simplifying the porting of applications to other databases.

The PHP Data Objects extension is currently included by default with installations of PHP 5.1. It is available for users of PHP 5.0 through PECL, but does not ship with the base package.

PDO uses features of PHP that were originally introduced in PHP 5.0. It is therefore not available for users of PHP 4.x and below.

Differences between PDO and the mysql extension

PHP Data Objects has a number of significant differences to the MySQL interface used by most PHP applications on PHP 4.x and below:

Object-orientation. While the mysql extension used a number of function calls that operated on a connection handle and result handles, the PDO extension has an object-oriented interface.

Database independence. The PDO extension, unlike the mysql extension, is designed to be compatible with multiple databases with little effort on the part of the user, provided standard SQL is used in all queries.

Connections to the database are made with a Data Source Name, or DSN. A DSN is a string that contains all of the information necessary to connect to a database, such as 'mysql:dbname=test_db'.

Intigration with (html,forms..etc)

Integrating PHP

"So," you say, "I now know the basics of this language. But, um... how do I use it?" I'm glad you asked. There are quite a few ways that PHP is used. You already know that you can call a script directly from a URL on your server. You can use PHP in more ways than that, though! Following are a few methods that PHP can be called.

Forms

Forms are, by far, the most common way of interacting with PHP. As we mentioned before, it is recommended that you have knowledge of HTML, and here is where we start using it. If you don't, just head to the HTML Wikibook for a refresher.

Form Setup

To create a form and point it to a PHP document, the HTML tag <form> is used and an action is specified as follows:

```
<form method="post" action="action.php">
<!-- Your form here -->
</form>
```

Once the user clicks "Submit", the form body is sent to the PHP script for processing. All fields in the form are stored in the variables $_GET or $_POST, depending on the method used to submit the form.

The difference between the GET and POST methods is that GET submits all the values in the URL, while POST submits values transparently through HTTP headers. A general rule of thumb is if you are submitting sensitive data, use POST. POST forms usually provide more security.

Remember $_GET and $_POST are superglobal arrays, meaning you can reference them anywhere in a PHP script. For example, you don't need to call global $_POST or global $_GET to use them inside functions.

Example

Let's look at an example of how you might do this.

```
<!-- Inside enterlogin.html -->
<html>
<head>
<title>Login</title>
</head>
<body>
<form method="post" action="login.php"> Please log in.<br/>
Username: <input name="username" type="text" /><br /> Password: <input name="password" type="password" /><br/>
<input name="submit" type="submit" />
</form>
</body>
</html>
```

This form would look like the following: Please log in. Username: …
Password: …
submit

And here's the script you'd use to process it (login.php):

```
<?php
// Inside enterlogin.html
if($_POST['username'] == "Spoom" && $_POST['password'] == "idontneednostinkingpassword")
{
echo("Welcome, Spoom.");
}
else
{
echo("You're not Spoom!");
}
?>
```

Let's take a look at that script a little closer. if($_POST['username'] == "Spoom" && $_POST['password'] == "idontneednostinkingpassword")

As you can see, $_POST is an array, with keys matching the names of each field in the form. For backward compatibility, you can also refer to them numerically, but you generally shouldn't as this method is much clearer. And that's basically it for how you use forms to submit data to PHP documents. It's that easy.

For More Information

PHP Manual: Dealing with Forms

PHP from the Command Line

Although PHP was originally created with the intent of being a web language, it can also be used for commandline scripting (although this is not common, because simpler tools such as the bash scripting are available).

Output

You can output to the console the same way you would output to a webpage, except that you have to remember that HTML isn't parsed (a surprisingly common error), and that you have to manually output newlines. The typical hello world program would look like this:

```
<?php
print "Hello World!\n";
?>
```

Notice the newline character at the end-of-line - newlines are useful for making your output look neat and readable.

Input

PHP has a few special files for you to read and write to the command line. These files include the stdin, stdout and stderr files. To access these files, you would open these files as if they were actual files using fopen, with the exception that you open them with the special php:// "protocol", like this:

```
$fp = fopen("php://stdin","r");
```

To read from the console, you can just read from stdin. No special redirection is needed to write to the console, but if you want to write to stderr, you can open it and write to it: $fp = fopen("php://stderr","w");

Bearing how to read input in mind, we can now construct a simple commandline script that asks the user for a username and a password to authenticate himself.

```
<?php
$fp = fopen("php://stdin","r");

print "Please authenticate yourself\n";
print "Username: ";
// rtrim to cut off the \n from the shell
$user = rtrim(fgets($fp, 1024));
print "Password: ";
// rtrim to cut off the \n from the shell
$pass = rtrim(fgets($fp, 1024));
if (($user=="someuser") && ($pass=="somepass")) {
print "Good user\n";
// ... do stuff ...
} else die("Bad user\n");
fclose($fp);
?>
```

PHP MySQL Prepared Statements

Prepared Statements and Bound Parameters

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

Compared to executing SQL statements directly, prepared statements have three main advantages:

- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
```

```
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

This function binds the parameters to the SQL query and tells the database what the parameters are. The "sss" argument lists the types of data that the parameters are. The s character tells mysql that the parameter is a string.

The argument may be one of four types:

- i - integer
- d - double
- s - string
- b - BLOB

We must have one of these for each parameter.By telling mysql what type of data to expect, we minimize the risk of SQL injections.

PHP libxml Functions

The libxml functions and constants are used together with SimpleXML, XSLT and DOM functions.

Function	Description
libxml_clear_errors()	Clears the libxml error buffer
libxml_disable_entity_loader()	Enables the ability to load external entities
libxml_get_errors()	Gets the errors from the the libxml error buffer
libxml_get_last_error()	Gets the last error from the the libxml error buffer
libxml_set_external_entity_loader()	Changes the default external entity loader
libxml_set_streams_context()	Sets the streams context for the next libxml document load or write
libxml_use_internal_errors()	Disables the standard libxml errors and enables user error handling