WEB PROGRAMMING

CSE VI SEM(CBCS)

UNIT I TO UNIT V

NOTES

PREPARED BY SANDEEP R,ASSISTANT PROFESSOR,CSE,MCET

SYLLABUS

UNIT-I

IntroductiontoWorldWideWeb, Web Browsers, Web Servers, UniformResourceLocators, HTTP.

HTML5: Introduction, Links, Images, Multimedia, Lists, Tables, Creating Forms, Styling Forms.

UNIT-II

Introduction to XML, XML document structure, Document Type Definition, Name spaces, XML Schemas, Display in graw XML documents, Displaying XML documents with CSS, XPath Basics, XSLT, XML Processors.

UNIT-III

Introduction to JavaScript: JavaScript and Forms Variables, Functions, Operators, Conditional Statements and Loops, Arrays DOM, Strings, Event and Event Handling, Java Script Closures. Introduction to Ajax: Pre-Ajax JavaScript Communication Techniques, XML HTTP Request Object, Data Formats, Security Concerns, User Interface Design for Ajax.

Introduction to Python: Objects and Methods Flow of Control, Dynamic Web Pages.

UNIT-IV

Java Servlets: Java Servlets and CGI Programming, Benefits of Java Servlet, Life Cycle of Java Servlet, Reading data from client, HTTP Request Header, HTTP Response Header, working with Cookies, Tracking Sessions. Java Server Pages: Introduction to JSP, JSP Tags, Variables and Objects, Methods, Control Statements, Loops, Request String, User Sessions, Session Object, Cookies.

UNIT-V

Introduction to PHP: Overview of PHP, General Syntactic Characteristics, Primitives, Operations, Expressions, Control Statements, Arrays, Functions, Pattern matching, Form handling, Files, Cookies, Session Tracking. Database access through Web: Architectures for Database Access-Database access with Perl-Database access with PHP-Database access with JDBC.

UNIT 1

Introduction to World Wide Web

The World Wide Web (www, W3) is an information space where documents and other web resources are identified by URIs, interlinked by hypertext links, and can be accessed via the Internet. It has become known simply as *the Web*. Hypertext documents are commonly called *web pages*, which are primarily text documents formatted and annotated with the Hypertext Markup Language (HTML). Webpages may contain links to images, video, and software components that are rendered to users of a web browser application, running on the user's computer, as coherent pages of multimedia content. Embedded hyperlinks permit users to navigate between web pages. When multiple web pages are published with a common theme or within a common domain name, the collection is usually called a *web site*.

Internet

The Internet is a huge collection of computers connected in a communications network. These computers are of every imaginable size, configuration, and manufacturer. In fact, some of the devices connected to the Internet—such as plotters and printers—are not computers at all.

The innovation that allows all of these diverse devices to communicate with each other is a single, low-level protocol: The Transmission Control Protocol/Internet Protocol (TCP/IP). TCP/IP became the standard for computer network connections in 1982.

It can be used directly to allow a program on one computer to communicate with a program on another computer via the Internet. In most cases, however, a higher-level protocol runs on top of TCP/IP.

Internet Protocol Address

The Internet Protocol (IP) address of a machine connected to the Internet is a unique 32-bit number. IP addresses usually are written as four 8-bit numbers, separated by periods.

The four parts are separately used by Internet-routing computers to decide where a message must go next to get to its destination. numbers, separated by periods.

The four parts are separately used by Internet-routing computers to decide where a message must go next to get to its destination. Although people nearly always type domain names into their browsers, the IP works just as well. For example, the IP for United Airlines (www.ual.com)

Domain Names

Because people have difficulty dealing with and remembering numbers, machines on the Internet also have textual names. These names begin with the name of the host machine, followed by progressively larger enclosing collections of machines, called domains.

There may be two, three, or more domain names. The first domain name, which appears immediately to the right of the host name, is the domain of which the host is a part.

The second domain name gives the domain of which the first domain is a part. The last domain name identifies the type of organization in which the host resides, which is the largest domain in the site's name

Web Browsers

A web browser is a software program that allows a user to locate, access, and display web pages. In common usage, a web browser is usually shortened to "browser." Browsers are used primarily for displaying and accessing websites on the internet, as well as other content created using languages such as Hypertext Markup Language (HTML) and Extensible Markup Language (XML).

Browsers translate web pages and websites delivered using Hypertext Transfer Protocol (HTTP) into human-readable content. They also have the ability to display other protocols and prefixes, such as secure HTTP (HTTPS), File Transfer Protocol (FTP), email handling (mailto:), and files (file:). In addition, most browsers also support external plug-ins required to display active content, such as in-page video, audio and game content.

Most commonly used web browsers are Google Chrome, Firefox, Internet Explorer, Opera, Safari, etc.

Web Servers

Web server is a computer where the web content is stored. Basically web server is used to host the web sites but there exists other web servers also such as gaming, storage, FTP, email etc.

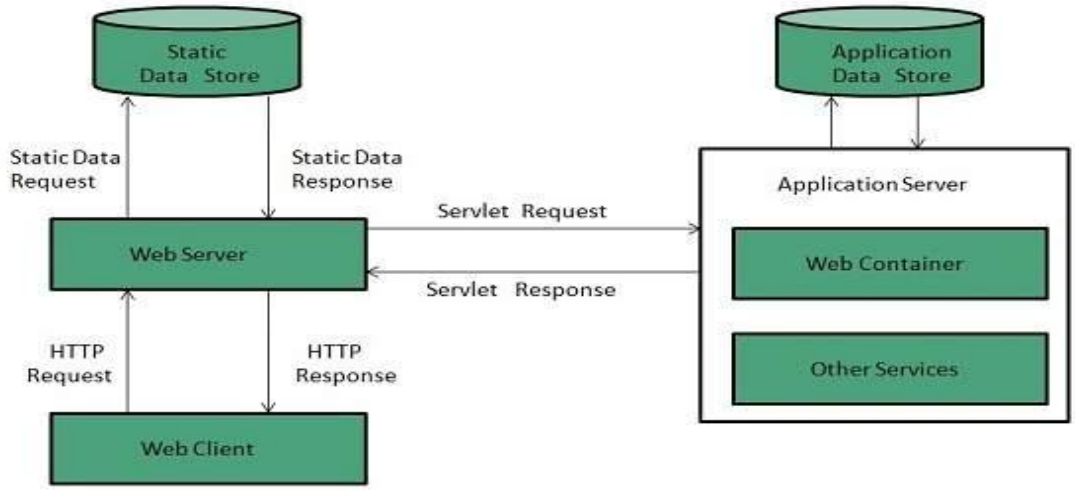
The most commonly used Web servers are Apache and Microsoft's Internet Information Server (IIS).

Web Server Working

Web server respond to the client request in either of the following two ways:

Sending the file to the client associated with the requested URL.

Generating response by invoking a script and communicating with database



Key Points

When client sends request for a web page, the web server search for the requested page if requested page is found then it will send it to client with an HTTP response.

If the requested web page is not found, web server will the send an HTTP response:Error 404 Not found.

If client has requested for some other resources then the web server will contact to the application server and data store to construct the HTTP response.

Architecture

Web Server Architecture follows the following two approaches:

Concurrent Approach

Single-Process-Event-Driven Approach.

Concurrent Approach

Concurrent approach allows the web server to handle multiple client requests at the same time. It can be achieved by following methods:

Multi-process

Multi-threaded

Hybrid method.

Uniform Resource Locator (URL)

Uniform (or universal) resource locators (URLs) are used to identify documents (resources) on the Internet. There are many different kinds of resources, identified by different forms of URLs.

URL Formats

All URLs have the same general format: scheme: object-address .The scheme is often a communications protocol. Common schemes include http, ftp, gopher, telnet, file, mailto, and news.

HTTP protocol supports the Web. This protocol is used to request and send eXtensible Hypertext Markup Language (XHTML) documents. In the case of HTTP, the form of the object address of a URL is as follows: //fully-qualified-domain-name/path-to-document .

URL Paths

The path to the document for the HTTP protocol is similar to a path to a file or directory in the file system of an operating system and is given by a sequence of directory names and a file name, all separated by whatever separator character the operating system uses. For UNIX servers, the path is specified with forward slashes; for Windows servers, it is specified with backward slashes.

The path in a URL can differ from a path to a file because a URL need not include all directories on the path. A path that includes all directories along the way is called a complete path. In most cases, the path to the document is relative to some base path that is specified in the configuration files of the server. Such paths are called partial paths.

E.g.: http://www.gumboco.com/files/f99/storefront.html

Multipurpose Internet Mail Extensions (MIME)

Multipurpose Internet Mail Extensions (MIME) is an Internet standard that helps extend the limited capabilities of email by allowing insertion of images, sounds and text in a message. It was proposed by Bell Communications in 1991, and the specification was originally defined in June 1992 for RFCs 1341 and 1342.

MIME was designed to extend the format of email to support non-ASCII characters, attachments other than text format, and message bodies which contain multiple parts. MIME describes the message content type and the type of encoding used with the help of headers. All manually composed and automated emails are transmitted through SMTP in MIME format. The association of Internet email with SMTP and MIME standards is such that the emails are sometimes referred to as SMTP/MIME email. The MIME standard defines the content types which are of prime importance in communication protocols like HTTP for the World Wide Web. The data are transmitted in the form of email messages through HTTP even though the data are not an email.

The features offered by MIME to email services are as follows:

Support for multiple attachments in a single message

Support for non-ASCII characters

Support for layouts, fonts and colors which are categorized as rich text.

Support for attachments which may contain executables, audio, images and video files, etc.

Support for unlimited message length.

MIME is extensible because it defines a method to register new content types and other MIME attribute values. The format of a message body is described by MIME using special header directives. This is done so that the email can be represented correctly by the client.

MIME Version: The presence of MIME Version generally indicates whether the message is MIME formatted. The value of the header is 1.0 and it is shown as MIME-Version: 1.0. The idea behind this was to create more advanced versions of MIME like 2.0 and so on.

Content-Type: This describes the data's Internet media type and the subtype. It may consist of a 'charset' parameter separated by a semicolon specifying the character set to be used. For example: Content-Type: Text/Plain.

Content-Transfer-Encoding: It specifies the encoding used in the message body.

Content-Description: Provides additional information about the content of the message.

Content-Disposition: Defines the name of the file and the attachment settings and uses the attribute 'filename'.

Hypertext Transfer Protocol (HTTP)

All Web communications transactions use the same protocol: The Hypertext Transfer Protocol (HTTP). HTTP consists of two phases: the request and the response.

Each HTTP communication (request or response) between a browser and a Web server consists of two parts: a header and a body. The header contains information about the communication; the body contains the data of the communication if there is any.

The Request Phase

The general form of an HTTP request is as follows:

1. HTTP method Domain part of the URL HTTP version

2. Header fields

3. Blank line

4. Message body

The following is an example of the first line of an HTTP request: GET /storefront.html HTTP/1.1

HTTP Request Methods

Method	Description
GET	Returns the contents of the specified document
HEAD	Returns the header information of the specified document
POST	Executes the specified document, using the enclosed data
PUT	Replaces the specified document with the enclosed data
DELETE	Deletes the specified document

GET and POST are the most frequently used. The format of a header field is the field name followed by a colon and the value of the field. There are four categories of header fields:

1. General: For general information, such as the date

2. Request: Included in request headers

3. Response: For response headers

4. Entity: Used in both request and response headers

The Response Phase

The general form of an HTTP response is as follows:

1. Status line
2. Response header fields
3. Blank line
4. Response body

The status line includes the HTTP version used, a three-digit status code for the response, and a short textual explanation of the status code. For example, most responses begin with the following: HTTP/1.1 200 OK

The general meanings of the five categories specified by these first digits are shown in Table.

First digits of HTTP status codes

First Digit	Category
1	Informational
2	Success
3	Redirection
4	Client error
5	Server error

One of the more common status codes is one user never want to see: 404 Not Found, which means the requested file could not be found.

HTML5

Introduction: HTML stands for Hyper Text Markup Language. It is used to design web pages using markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. Markup language is used to define the text document within tag which defines the structure of web pages. HTML 5 is the fifth and current version of HTML. It has improved the markup available for documents and has introduced application programming interfaces(API) and Document Object Model(DOM).

Features:

It has introduced new multimedia features which supports audio and video controls by using <audio> and <video> tags.

There are new graphics elements including vector graphics and tags.

Enrich semantic content by including <header> <footer>, <article>, <section> and <figure> are added.

Drag and Drop- The user can grab an object and drag it further dropping it on a new location.

Geo-location services- It helps to locate the geographical location of a client.

Web storage facility which provides web application methods to store data on web browser.

Uses SQL database to store data offline.

Allows to draw various shapes like triangle, rectangle, circle, etc.

Capable of handling incorrect syntax.

Easy DOCTYPE declaration i.e. <!doctype html>

Easy character encoding i.e. <meta charset="UTF-8″>

Advantages:

All browsers supported.

More device friendly.

Easy to use and implement.

HTML 5 in integration with CSS, JavaScript, etc can help build beautiful websites.

Disadvantages:

Long codes have to be written which is time consuming.

Only modern browsers support it.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML 5</title>
<style>
h1
{
font-size:50px;
}
</style>
</head>
<body>
<h1>welcome to HTML5</h1>
</body>
</html>
```

Removed elements from HTML 5: There are many elements which are depreciated from HTML 5 are listed below:

REMOVED ELEMENTS	USE INSTEAD ELEMENTS
<acronym>	<abbr>
<applet>	<object>
<basefont>	CSS

REMOVED ELEMENTS	USE INSTEAD ELEMENTS
<big>	CSS
	CSS
<dir>	
<font>	CSS
<frame>	
<frameset>	
<noframes>	
<isindex>	
<strike>	CSS, <s> or <del>
<tt>	CSS

New Added Elements in HTML 5:

<article>: The <article> tag is used to represent an article. More specifically, the content within the <article> tag is independent from the other content of the site (even though it can be related).

<aside>: The <aside> tag is used to describe the main object of the web page in a shorter way like a highlighter. It basically identifies the content that is related to the primary content of the web page but does not constitute the main intent of the primary page. The <aside> tag contains mainly author information, links, related content and so on.

<figcaption>: The <figurecaption> tag in HTML is used to set a caption to the figure element in a document.

<figure>: The <figure> tag in HTML is used to add self-contained content like illustrations, diagrams, photos or codes listing in a document. It is related to main flow but it can be used in any position of a document and the figure goes with the flow of the document and if remove it then it should not affect the flow of the document.

<header>: It contains the section heading as well as other content, such as a navigation links, table of contents, etc.

<footer>: The <footer> tag in HTML is used to define a footer of HTML document. This section contains the footer information (author information, copyright information, carriers etc). The footer tag are used within body tag. The <footer> tag is new in the HTML 5. The footer elements require a start tag as well as an end tag.

<main>: Delineates the main content of the body of a document or web app.

<mark>: The <mark> tag in HTML is used to define the marked text. It is used to highlight the part of the text in the paragraph.

<nav>: The <nav> tag is used to declaring the navigational section in HTML documents. Websites typically have sections dedicated to navigational links, which enables user to navigate the site. These links can be placed inside a nav tag.

<section>: It demarcates a thematic grouping of content.

<details>: The <details> tag is used for the content/information which is initially hidden but could be displayed if the user wishes to see it. This tag is used to create interactive widget which user can open or close it. The content of details tag is visible when open the set attributes.

<summary>: The <summary> tag in HTML is used to define a summary for the <details> element. The <summary> element is used along with the <details> element and provides a summary visible to the user. When the summary is clicked by the user, the content placed inside the <details> element becomes visible which was previously hidden. The <summary> tag was added in HTMl 5. The <summary> tag requires both starting and ending tag.

<time>: The <time> tag is used to display the human-readable data/time. It can also be used to encode dates and times in a machine-readable form. The main advantage for users is that they can offer to add birthday reminders or scheduled events in their calender's and search engines can produce smarter search results.

<bdi>: The <bdi> tag refers to the Bi-Directional Isolation. It differentiate a text from other text that may be formatted in different direction. This tag is used when a user generated text with an unknown directions.

<wbr>: The <wbr> tag in HTML stands for word break opportunity and is used to define the position within the text which is treated as a line break by the browser. It is mostly used when the used word is too long and there are chances that the browser may break lines at the wrong place for fitting the text.

<datalist>: The <datalist> tag is used to provide autocomplete feature in the HTML files. It can be used with input tag, so that users can easily fill the data in the forms using select the data.

<keygen>: The <keygen> tag in HTML is used to specify a key-pair generator field in a form. The purpose of <keygen> element is to provide a secure way to authenticate users. When a from is submitted then two keys are generated, private key and public key. The private key stored locally, and the public key is sent to the server. The public key is used to generate client certificate to authenticate user for future.

<output>: The <output> tag in HTML is used to represent the result of a calculation performed by the client-side script such as JavaScript.

<progress>: It is used to represent the progress of a task. It is also define that how much work is done and how much is left to download a things. It is not used to represent the disk space or relevant query.

<svg>: It is the Scalable Vector Graphics.

<canvas>: The <canvas> tag in HTML is used to draw graphics on web page using JavaScript. It can be used to draw paths, boxes, texts, gradient and adding images. By default it does not contains border and text.

<audio>: It defines the music or audio content.

<embed>: Defines containers for external applications (usually a video player).

<source>: It defines the sources for <video> and <audio>.

<track>: It defines the tracks for <video> and <audio>.

<video>: It defines the video content.

1.Hyperlinks is one of the very core features of HTML, they enable you to jump from one webpage to another.

2.The very idea of *World Wide Web* is built around Hyperlinks.All day to day activities like Browsing, Surfing, Downloads depends upon links.

Types of Links:

Header Links: using <link> element within the <head> element.

Anchor Links: using anchor element<a> , within <body> element

We will study in detail about ANCHOR LINKS <a> in this lesson.

How to make links in html?

1.Any text can be transformed into a hyperlink by encapsulating it within anchor tag <a> .

2.The attribute href contains the URL of the webdocument,to which the clickable text links.

Syntax:

 Related text.

<!-- Example-->

 Tutorials Park.

Example of a Link:

<!DOCTYPE HTML>

<html>

<body>

Tutorials Park

is the most comprehensive web development website on the internet.

</body>

</html>

Commonly used Attributes of a Link.

href: Defines the URL or the location to which the link is created.

title: Title contains a short descriptive text related to the link,such as authors, description etc.Hover the tool tip over the link to see the title.

id : To assign an unique identifier to the link.Two elements on same page cannot have the same id.

class: To assign an identifier to an element,but multiple elements can have the same class name unlike attribute id.

HTML Link Target

Link element attribute "target" denotes the target of the browsing context(i.e Tab, new Window, iframe) in which the link should open.

Target attribute Values:

blank: To open the webpage in a new Tab or Window.

self: To open the clicked document in the frame as the current one.

parent: To open the Web document in the parent frameset of the current frame.If no parent available then it behaves as _self.

top: To open the web document in the full window removing all other frames.

<frame>: To open the webdocument in the specified or named frame

EX:

Follow us on facebook for new updates.

(opens in new Window)

Absolute and Relative Links

1. Absolute link URLs contain entire Destination address, including the Protocol(http,https,ftp etc) and domain name.The Url doesn't change no matter where the webpage it appears on is kept. Hence, its called Absolute Links.

2.Relative Links do not contain the entire Destination address.Its address is relative w.r.t to the Document on which is appears.

a.The Url http://www.tutorialspark/html/html5_links is an absolute link.

b.The Url ../html/html5_links is a relative link w.r.t to a document on the same website.

c.Relative links will appear broken if the address of the webpage on which the link appears changes.

EX:

p> This one here

is an Absolute hyperlink.

 While this one here

is a relative hyperlink.

Images

Images can improve the design and the appearance of a web page. The image is inserted into the display of the document by the browser.

The <IMG> Tag

Building on this idea of attributes I can introduce you to the IMG tag. To insert an image in a page you can use the following HTML snippet – I will explain each attribute in turn.

Image Formats

The two most common methods of representing images are the Graphic Interchange Format (GIF) and the Joint Photographic Experts Group format. Most contemporary browsers can render images in either of these two formats. Files in both formats are compressed to reduce storage needs and provide faster transfer over the Internet.

GIF uses 8-bit color representations for pixels, allowing a pixel to have 256 different colors. Files containing GIF images use the .gif (or .GIF) extension on their names. GIF images can be made to appear transparent.

The JPEG format uses 24-bit color representations for pixels, which allows JPEG images to include more than 16 million different colors. Files that store JPEG images use the .jpg (or .JPG or .jpeg) extension on their names.

Portable Network Graphics (PNG) is a good replacement for both GIF and JPEG because it has the best characteristics of each (the possibility of transparency, as provided by GIF, and the same large number of colors as JPEG). One drawback of PNG is that, because its compression algorithm does not sacrifice picture clarity.

The <img /> tag

In HTML, images are defined with the <img> tag.

The src Attribute

This stands for ‘source’ – i.e. where is your image file? This can be an image file on your own website, or on another website, or on a CDN (Content Delivery Network).

If you are embedding an image from your own site, you can specify the full URL to the image or, as in this example, a path relative to the root of your website. If you are using files on your own computer with these HTML5 tutorials, then you can either use the full folder path to the image, or you can use a path relative to where your .html file is.

This makes your options something like this:

src="https://www.domain.com/images/my-company-logo.png" – the full URL to the image

src="https://www.not-my-website.com/images/their-company-logo.png" – a URL to an external site's images

src="images/logos/my-company-logo.png" – take the location of my .html file, find a folder in the same location called 'images', then look for a folder in 'images' called 'logos', then find 'my-company-logo.png'

Valid File Formats

There are quite a large array of image formats these days. However, the most common ones used in an <img> tag are

GIF: hsa a limit to the number of colours you can use, but supports animation and transparency

JPG or JPEG: no limit on the number of colours but does not support transparency.

PNG: creates a high quality image, which can lead to a high file size, but supports transparency.

There are other formats but these are enough to get you started.

Embedding Images From Other Websites

Before you consider doing this, please make sure that, first and foremost, **you have the right** to use that image. Reaching out to the site to ask permission would be a good starting point.

Another consideration is that their server may prevent you doing this. Embedding another website's images is called **hotlinking** and uses another person's bandwidth. For that reason always look to host the images on your own web hosting, or subscribe to a CDN.

The width and height Attributes

Sounds obvious, but this sets the height and width of your image. This is measured in , but you can equally specify a for the width element.

If you do not specify a height and a width, your browser will attempt to load the image and the size it has been created. I say 'attempt' because CSS can limit how large an image can be. Also another limiting factors might be the width of a table cell. So you are better to take control of your page and define the height and width of your image.

The alt Attribute

alt is short for 'alternative' and should be used as a text description of your image. This serves a couple of purposes:

Accessibility: Providing a text description of your image will help those who browse either with images turned off, or with screen readers (for those with eyesight challenges) to understand what that image is.

SEO: Search engines use the alt attribute to help determine the nature of the image and helps those images to show up for relevant queries in image search results. Carefully implementing this can also influence how the page the image is on to rank for relevant terms.

The title Attribute

The title attribute will display a 'tooltip' when someone hovers their cursor over the image. Again this should be descriptive, but best to keep this shorter than the alt attribute as tooltips tend to flash up for only a few seconds.

The align Attribute

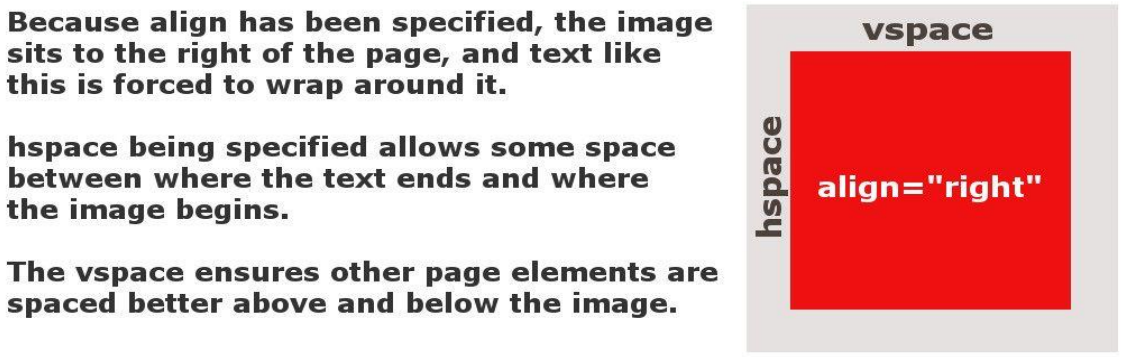
The align attribute determines where on your page the image will sit. If you do not specify an align attribute, then **it will align to the left of your page** but your text **will not wrap around your image**.

You can specify **left, right, or center** but support for center is patchy and cannot really be relied on. I will cover in the CSS Tutorials how to force this center alignment.

If you specify left or right then your image will sit on the appropriate side of your page and **your text will wrap around** the image, at least within the limit of the size of the browser window loading the page.

hspace and vspace

These two attributes give your image space, so that there is a gap between the image and the elements around it. Here's a demonstration of how align, vspace and hspace work:



ultimedia

Multimedia on the web is sound, music, videos, movies, and animations.Multimedia comes in many different formats. It can be almost anything you can hear or see.Examples: Images, music, sound, videos, records, films, animations, and more.

Web pages often contain multimedia elements of different types and formats.The most common way to discover the type of a file, is to look at the file extension.The first web browsers had support for text only, limited to a single font in a single color.

HTML5 has introduced two new multimedia tags, AUDIO and VIDEO, for displaying the audio and video streams on a Web page.

You can play the multimedia files, which are stored in your local computer, on the Web page by specifying their location. The src attribute is used to specify the multimedia file to play it on the Web page.

If the Web browser does not support AUDIO and VIDEO tags, then the text defined between the starting and the closing tags of these tags are displayed on the Web page.

Attributes of AUDIO Tag

The AUDIO tag of HTML5 supports only three audio file formats i.e. .oog, .mp3, .wav

Following table shows the attributes of the AUDIO tag

Attribute	Description
autoplay	Plays the audio file as soon as the Web page loads
controls	Displays the controls on the Web page, such as play and pause buttons
loop	Replays the audio file
preload	Specifies whether the audio file is preloaded on the Web page or not
src	Provides the location of the audio file to play

Attribute	Description
audio	Controls the default state of the video's audio channel
autoplay	Plays the audio file as soon as the Web page loads
controls	Displays the controls on a Web page, such as play and pause buttons
height	Specifies the height of the VIDEO tag
loop	Replays the video file
preload	Specifies whether the video file is preloaded on the Web page or not
poster	Provides an image to be displayed when the video file is not available
src	Provides the location of the video file to play
width	Specifies the width of the VIDEO tag

You can use the VIDEO tag to display a video file on the Web page. The VIDEO tag supports the .gov and .mp4 file formats.

Following table describes attributes of the VIDEO tag

You can also use the SOURCE tag within the opening and the closing tags of the VIDEO tag to provide the source of the video file.

The SOURCE tag is used in a situation when the location of the video file is not confirmed. In this case, the VIDEO tag plays the first video file located in the specified path. The following code snippet shows the use of the VIDEO tag :

<VIDEO src="video.ogv" autoplay="true" loop="3" controls>

</VIDEO>

In the above code snippet, we have defined a video.ogv file in the src attribute. We have also set the autoplay attribute to true, which implies that the video file start playing as soon as the Web page loads. the loop attribute is set to 3, which implies that the video file will be played three times. In addition, the controls attribute displays the controls on the video player.

Hypertext Links

A hypertext link in an XHTML document, which we simply call a link here, acts as a pointer to some particular place in some Web resource. That resource can be an XHTML document anywhere on the Web, or it may be the document currently being displayed. Without links, Web documents would be boring and tedious to read.

Most Web sites consist of many different documents, all logically linked. Therefore, links are essential to building an interesting Web site.

When you move the mouse over a link, the mouse arrow will turn into a little hand. A link does not have to be text. It can be an image or any other HTML element.

Syntax

Links are specified in an attribute of an anchor tag (), which is an inline tag. The anchor tag that specifies a link is called the source of that link. The document whose address is specified in a link is called the target of that link.

Visit our HTML tutorial

By default, a link will appear like this (in all browsers):

-An unvisited link is underlined and blue

-A visited link is underlined and purple

-An active link is underlined and red

The target attribute specifies where to open the linked document.The target attribute can have one of the following values:

_blank - Opens the linked document in a new window or tab

_self - Opens the linked document in the same window/tab as it was clicked (this is default)

_parent - Opens the linked document in the parent frame

_top - Opens the linked document in the full body of the window

framename - Opens the linked document in a named frame

Lists

List are used to group related contents together in a structured manner making content easy to read and understand.

Types of Lists

Ordered List: To group a set of related items in a specific numbered order.

Unordered List: To group a set of related items in no specific order

Definitions List: To group a set of related terms and their definitions.

Nested List: To create a list within another list(i.e nested).

Ordered List

1. An Ordered list has each item numbered, this is useful to provide sequential instructions.Ordered list is used extensively in formal documents.

2. The Ordered element < ol > encapsulates the complete list, and list element < li > encapsulates each of list items .

List Example

```
<ol>
<li>Macbook Air</li>
<li>iPhone</li>
<li>iPad</li>
<li>iMac</li>
</ol>
```

Unordered List

1. Unordered list is similar to ordered list but the numerals are replaced with bullet points.

2. Unordered list is created within unordered tag < ul > and similar to ordered list ,list items are placed within list tag < li > .

3. Unordered list attribute " list-style-type" property can be used to replace bullet points with disc, circular and square bullets.

List Example

```
<ul>                          <!-- Unordered Element */-->
<li>BMW.</li>
<li>MERCEDES.</li>            <!--list tag -->
<li>PORSCHE.</li>
<li>AUDI.</li>
</ul>
```

Definition List

1. The Structure of definiton list is a term followed by its description or definition.

2. Its created with a Definition list tag < dl > encapsulating the complete list .

3. Within each < dl > tag you have a pair of definition term <dt> and definition description element <dd>.

4. <dt> contains definition term, while <dd> contains the definition or description.

List Example

```
<dl>
<dt>iPAD</dt>
<dd>- A family of Tablets by APPLE Inc.</dd>
<dt>iPhone</dt>
<dd>- A Series of Smart Phones by Apple.</dd>
<dd>- First Launched in USA. </dd>
<dt>Macbook Pro</dt>
<dd>- Macintosh Portable computers from Apple.</dd>
<dt>Windows</dt>
</dl>
```

Nested List

1. Nested list is created by inserting a list within a list.The Sole idea behind this is to create a sublist for a main list item .

3. The Sublist list is put inside the <li> element of the main list .

Example of Nested List

```
<ul>
<li>Linux</li>
<li>Windows</li>   // main list item.
<ul>
<li>Windows 95.</li>
<li>Windows 98.</li>//sublist.
<li>Windows XP.</li>
<li>Windows 7.</li>
</ul>
</li>
<li> Mac OS X</li>
</ul>
```

ordered List: Starting with user defined order

```
<ol start="4">
<li>Macbook Air</li>
<li>iPhone</li>
<li>iPad</li>
<li>iMac</li>
</ol>
```

Tables

In HTML Table will be created by using <table>table data goes here..</table> tag.

We know that table contains Rows and Columns, those are defined with tr and td.

<tr> stands for Table Row which is used to make a Row.

<td> stands for Table Data that is used to make a Column.

Table heading can be defined by using <th>Name</th>

Cellpadding and Cellspacing is used to adjust the white space in table cell.

Cellspacing defines the width of the border. cellspacing="0" cellpadding="15"

Cellpadding represents the distance between cell borders and the content within.

<caption> Books Information</caption> tag will serve as a title and show at the top of the table.

Example table

```
<table>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
</table>
```

UNIT II

What is xml

- **Xml** (eXtensible Markup Language) is a mark up language.
- XML is designed to store and transport data.
- Xml was released in late 90's. it was created to provide an easy to use and store self describing data.
- XML became a W3C Recommendation on February 10, 1998.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

What is mark-up language

A **mark up language** is a modern system for highlight or underline a document.
Students often underline or highlight a passage to revise easily, same in the sense of modern mark up language highlighting or underlining is replaced by tags.

Why xml

Platform Independent and Language Independent: The main benefit of xml is that you can use it to take data from a program like Microsoft SQL, convert it into XML then share that XML with other programs and platforms. You can communicate between two platforms which are generally very difficult.
The main thing which makes XML truly powerful is its international acceptance. Many corporation use XML interfaces for databases, programming, office application mobile phones and more. It is due to its platform independent feature.

Features and Advantages of XML

XML is widely used in the era of web development. It is also used to simplify data storage and data sharing.

The main features or advantages of XML are given below.

1) XML separates data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.
With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

2) XML simplifies data sharing

In the real world, computer systems and databases contain data in incompatible formats.
XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.
This makes it much easier to create data that can be shared by different applications.

3) XML simplifies data transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.
Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

4) XML simplifies Platform change

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.
XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

5) XML increases data availability

Different applications can access your data, not only in HTML pages, but also from XML data sources.
With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

6) XML can be used to create new internet languages

A lot of new Internet languages are created with XML.
Here are some examples:

- **XHTML**
- **WSDL** for describing available web services
- **WAP** and **WML** as markup languages for handheld devices
- **RSS** languages for news feeds
- **RDF** and **OWL** for describing resources and ontology
- **SMIL** for describing multimedia for the web

XML Example

XML documents create a hierarchical structure looks like a tree so it is known as XML Tree that starts at "the root" and branches to "the leaves".
Syntax:

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Example of XML Document

XML documents uses a self-describing and simple syntax:

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

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

```
<note>
```

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

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

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

```
</note>
```

XML: Books.xml

```
<bookstore>
<book category="COOKING">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
```

```
<price>30.00</price>
</book>
<book category="CHILDREN">
<title lang="en">Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="WEB">
<title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

XML: Emails.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<emails>
<email>
  <to>Vimal</to>
  <from>Sonoo</from>
  <heading>Hello</heading>
  <body>Hello brother, how are you!</body>
</email>
<email>
  <to>Peter</to>
  <from>Jack</from>
  <heading>Birth day wish</heading>
  <body>Happy birth day Tom!</body>
</email>
<email>
  <to>James</to>
  <from>Jaclin</from>
  <heading>Morning walk</heading>
  <body>Please start morning walk to stay fit!</body>
```

```
</email>
<email>
  <to>Kartik</to>
  <from>Kumar</from>
  <heading>Health Tips</heading>
  <body>Smoking is injurious to health!</body>
</email>
</emails>
```

No.	Technology	Meaning	Description
1)	XHTML	Extensible html	It is a clearer and stricter version of XML. It belongs to the family of XML markup languages. It was developed to make html more extensible and increase inter-operability with other data.
2)	XML DOM	XML document object model	It is a standard document model that is used to access and manipulate XML. It defines the XML file in tree structure.
3)	XSL it contain three parts: i) XSLT (xsl transform) ii) XSL iii)XPath	Extensible style sheet language	i) It transforms XML into other formats, like html. ii) It is used for formatting XML to screen, paper etc. iii) It is a language to navigate XML documents.
4)	XQuery	XML query language	It is a XML based language which is used to query XML based data.
5)	DTD	Document type definition	It is an standard which is used to define the legal elements in an XML document.
6)	XSD	XML schema definition	It is an XML based alternative to dtd. It is used to describe the structure of an XML document.
7)	XLink	XML linking	xlink stands for XML linking language. This is a

		language	language for creating hyperlinks (external and internal links) in XML documents.
8)	XPointer	XML pointer language	It is a system for addressing components of XML based internet media. It allows the xlink hyperlinks to point to more specific parts in the XML document.
9)	SOAP	Simple object access protocol	It is an acronym stands simple object access protocol. It is XML based protocol to let applications exchange information over http. in simple words you can say that it is protocol used for accessing web services.
10)	WSDL	web services description languages	It is an XML based language to describe web services. It also describes the functionality offered by a web service.
11)	RDF	Resource description framework	RDF is an XML based language to describe web resources. It is a standard model for data interchange on the web. It is used to describe the title, author, content and copyright information of a web page.
12)	SVG	Scalable vector graphics	It is an XML based vector image format for two-dimensional images. It defines graphics in XML format. It also supports animation.
13)	RSS	Really simple syndication	RSS is a XML-based format to handle web content syndication. It is used for fast browsing for news and updates. It is generally used for news like sites.

XML Attributes

XML elements can have attributes. By the use of attributes we can add the information about the element.
XML attributes enhance the properties of the elements.

```
<book publisher="Tata McGraw Hill"></book>
```

Here, book is the element and publisher is the attribute.

Metadata should be stored as attribute and data should be stored as element.

```
<book>
```

```
<book category="computer">
<author> A & B </author>
</book>
```

Difference between attribute and sub-element

1st way:

```
<book publisher="Tata McGraw Hill"> </book>
```

2nd way:

```
<book>
<publisher> Tata McGraw Hill </publisher>
</book>
```

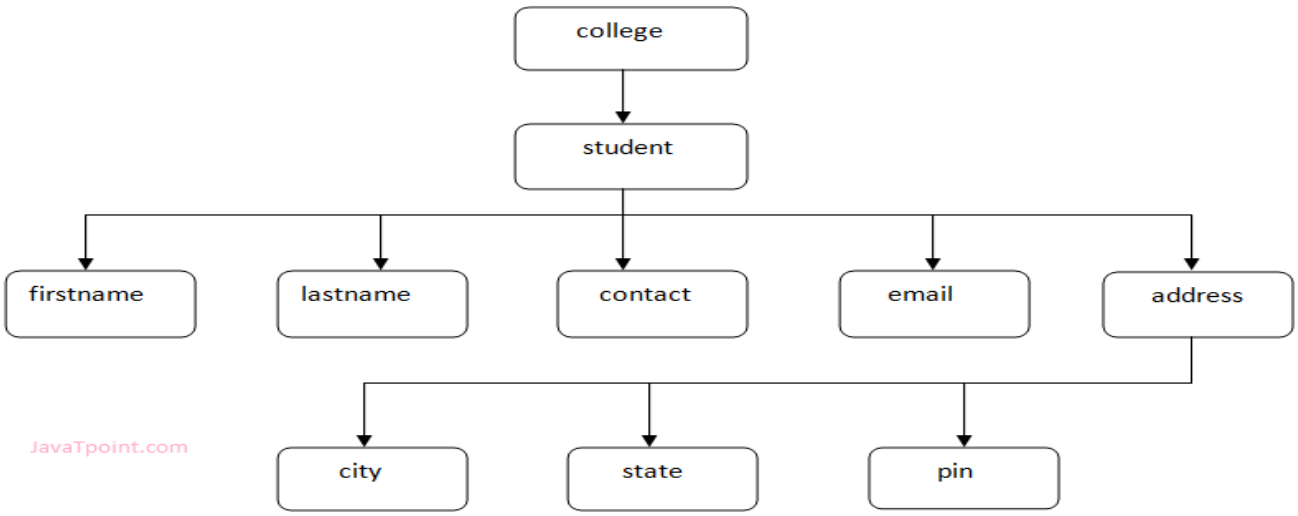
syntax ***An XML comment should be written as:*** ***<!-- Write your comment-->***

1. Don't use a comment before an XML declaration.
2. You can use a comment anywhere in XML document except within attribute value.
3. Don't nest a comment inside the other comment.

XML Tree Structure

An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.

```
<?xml version="1.0"?>
<college>
  <student>
    <firstname>Tamanna</firstname>
    <lastname>Bhatia</lastname>
    <contact>09990449935</contact>
    <email>tammanabhatia@abc.com</email>
    <address>
       <city>Ghaziabad</city>
       <state>Uttar Pradesh</state>
       <pin>201007</pin>
    </address>
  </student>
</college>
```



XML Validation

A well formed XML document can be validated against DTD or Schema.
A well-formed XML document is an XML document with correct syntax. It is very necessary to know about valid XML document before knowing XML validation.

Valid XML document

It must be well formed (satisfy all the basic syntax condition)
It should be behave according to predefined DTD or XML schema

Rules for well formed XML

- It must begin with the XML declaration.
- It must have one unique root element.
- All start tags of XML documents must match end tags.
- XML tags are case sensitive.
- All elements must be closed.
- All elements must be properly nested.
- All attributes values must be quoted.
- XML entities must be used for special characters.

Document Type Definition (DTD)

What is a DTD?
A DTD is a Document Type Definition. A DTD defines the structure and the legal elements and attributes of an XML document.
Why Use a DTD?
With a DTD, independent groups of people can agree on a standard DTD for interchanging data.

An application can use a DTD to verify that XML data is valid.
In a DTD, elements are declared with an ELEMENT declaration.
Declaring Elements
In a DTD, XML elements are declared with the following syntax:
<!ELEMENT element-name category>
or
<!ELEMENT element-name (element-content)>
Empty Elements
Empty elements are declared with the category keyword EMPTY:
<!ELEMENT element-name EMPTY>
Example:
<!ELEMENT br EMPTY>
XML example:

Elements with Parsed Character Data
Elements with only parsed character data are declared with #PCDATA inside parentheses:
<!ELEMENT element-name (#PCDATA)>
Example:
<!ELEMENT from (#PCDATA)>
Elements with any Contents
Elements declared with the category keyword ANY, can contain any combination of parsable data:
<!ELEMENT element-name ANY>
Example:
<!ELEMENT note ANY>
Elements with Children (sequences)
Elements with one or more children are declared with the name of the children elements inside parentheses:
<!ELEMENT element-name (child1)>
or
<!ELEMENT element-name (child1,child2,...)>
Example:
<!ELEMENT note (to,from,heading,body)>
Declaring Minimum One Occurrence of an Element
<!ELEMENT element-name (child-name+)>
Example:
<!ELEMENT note (message+)>
Declaring Zero or More Occurrences of an Element
<!ELEMENT element-name (child-name*)>
Example:
<!ELEMENT note (message*)>
Declaring Zero or One Occurrences of an Element
<!ELEMENT element-name (child-name?)>
Example:
<!ELEMENT note (message?)>
Declaring either/or Content

<!ELEMENT note (to,from,header,(message|body))>
Declaring Mixed Content
<!ELEMENT note (#PCDATA|to|from|header|message)*>
DTD - Attributes
In a DTD, attributes are declared with an ATTLIST declaration.

Declaring Attributes
An attribute declaration has the following syntax:
<!ATTLIST element-name attribute-name attribute-type attribute-value>
DTD example:
<!ATTLIST payment type CDATA "check">
XML example:
<payment type="check" />

The **attribute-type** can be one of the following:

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

The **attribute-value** can be one of the following:

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

A Default Attribute Value
DTD:

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

Valid XML:

```
<square width="100" />
```

In the example above, the "square" element is defined to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

#REQUIRED
Syntax

```
<!ATTLIST element-name attribute-name attribute-type #REQUIRED>
```

Example
DTD:

```
<!ATTLIST person number CDATA #REQUIRED>
```

Valid XML:

```
<person number="5677" />
```

Invalid XML:

```
<person />
```

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

#IMPLIED
Syntax

```
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
```

Example
DTD:

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

Valid XML:

```
<contact fax="555-667788" />
```

Valid XML:
<contact />
Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

#FIXED
Syntax
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
Example
DTD:
<!ATTLIST sender company CDATA #FIXED "Microsoft">

Valid XML:
<sender company="Microsoft" />

Invalid XML:
<sender company="W3Schools" />
Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Enumerated Attribute Values
Syntax
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
Example
DTD:
<!ATTLIST payment type (check|cash) "cash">

XML example:
<payment type="check" />
or
<payment type="cash" />
Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

XML document with an internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

The DTD above is interpreted like this:

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element must contain four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
- **!ELEMENT body** defines the body element to be of type "#PCDATA"

An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

The DTD above is interpreted like this:

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element must contain four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
- **!ELEMENT body** defines the body element to be of type "#PCDATA"

An External DTD Declaration

If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
```

```
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

And here is the file "note.dtd", which contains the DTD:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

XML DTD with entity declaration

A doctype declaration can also define special strings that can be used in the XML file.
An entity has three parts:

1. An ampersand (&)
2. An entity name
3. A semicolon (;)

Syntax to declare entity:<!ENTITY entity-name "entity-value">

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

XML Namespaces

XML **Namespace** is used *to avoid element name conflict* in XML document.

XML Namespace Declaration

An XML namespace is declared using the reserved XML attribute. This attribute name must be started with "xmlns".
Let's see the XML namespace syntax: <**element** xmlns:name = "URL">
Here, namespace starts with keyword "**xmlns**". The word **name** is a namespace prefix.
The **URL** is a namespace identifier.

Let's see the example of XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<cont:contact xmlns:cont="http://sssit.org/contact-us">
<cont:name>Vimal Jaiswal</cont:name>
<cont:company>SSSIT.org</cont:company>
<cont:phone>(0120) 425-6464</cont:phone>
</cont:contact>
```

Namespace Prefix: cont
Namespace Identifier: http://sssit.org/contact-us
It specifies that the element name and attribute names with cont prefix belongs to http://sssit.org/contact-us name space.
In XML, elements name are defined by the developer so there is a chance to conflict in name of the elements. To avoid these types of confliction we use XML Namespaces. We can say that XML Namespaces provide a method to avoid element name conflict.
Generally these conflict occurs when we try to mix XML documents from different XML application.

Table1:

```
<table>
  <tr>
    <td>Aries</td>
    <td>Bingo</td>
  </tr>
</table>
```

Table2:

```
<table>
<tr>
  <name>Computer table</name>
  <width>80</width>
  </tr>
</table>
```

If you add these both XML fragments together, there would be a name conflict because both have <table< element. Although they have different name and meaning.

XML Schema

What is XML schema

XML schema is a language which is used for expressing constraint about XML documents.
There are so many schema languages which are used now a days for example Relax- NG and XSD (XML schema definition).
An XML schema is used to define the structure of an XML document. It is like DTD but provides more control on XML structure.
Why Learn XML Schema?
In the XML world, hundreds of standardized XML formats are in daily use.
Many of these XML standards are defined by XML Schemas.
XML Schema is an XML-based (and more powerful) alternative to DTD.
XML Schemas use XML Syntax
Another great strength about XML Schemas is that they are written in XML.

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas are extensible, because they are written in XML.
With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types

- Reference multiple schemas in the same document

The <schema> Element

The <schema> element is the root element of every XML Schema:

```
<?xml version="1.0"?>

<xs:schema>
...
...
</xs:schema>
```

XSD Simple Elements

XML Schemas define the elements of your XML files.

A simple element is an XML element that contains only text. It cannot contain any other elements or attributes

Defining a Simple Element

The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

where xxx is the name of the element and yyy is the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example

Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

And here are the corresponding simple element definitions:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Default and Fixed Values for Simple Elements

Simple elements may have a default value OR a fixed value specified.

A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "red":

```
<xs:element name="color" type="xs:string" default="red"/>
```

A fixed value is also automatically assigned to the element, and you cannot specify another value.

In the following example the fixed value is "red":

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

XSD Attributes

All attributes are declared as simple types.

What is an Attribute?
Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.
How to Define an Attribute?
The syntax for defining an attribute is:

```
<xs:attribute name="xxx" type="yyy"/>
```

where xxx is the name of the attribute and yyy specifies the data type of the attribute.
XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example
Here is an XML element with an attribute:

```
<lastname lang="EN">Smith</lastname>
```

And here is the corresponding attribute definition:

```
<xs:attribute name="lang" type="xs:string"/>
```

What is a Complex Element?

A complex element is an XML element that contains other elements and/or attributes.
There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Note: Each of these elements may contain attributes as well!
Examples of Complex Elements
A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

A complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

A complex XML element, "food", which contains only text:

```
<food type="dessert">Ice cream</food>
```

A complex XML element, "description", which contains both elements and text:

```
<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

How to Define a Complex Element
Look at this complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

We can define a complex element in an XML Schema two different ways:
1. The "employee" element can be declared directly by naming the element, like this:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

If you use the method described above, only the "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared. You will learn more about indicators in the XSD Indicators chapter.
2. The "employee" element can have a type attribute that refers to the name of the complex type to use:

```
<xs:element name="employee" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Complex Empty Elements
An empty XML element:

```
<product prodid="1345" />
```

String Data Type
The string data type can contain characters, line feeds, carriage returns, and tab characters.
The following is an example of a string declaration in a schema:

```
<xs:element name="customer" type="xs:string"/>
```

Date Data Type
The date data type is used to specify a date.
The date is specified in the following form "YYYY-MM-DD" where:

- YYYY indicates the year
- MM indicates the month
- DD indicates the day

Note: All components are required!
The following is an example of a date declaration in a schema:

<xs:element name="start" type="xs:date"/>

Decimal Data Type

The decimal data type is used to specify a numeric value.

The following is an example of a decimal declaration in a schema:

<xs:element name="prize" type="xs:decimal"/>

Boolean Data Type

The boolean data type is used to specify a true or false value.

The following is an example of a boolean declaration in a schema:

<xs:attribute name="disabled" type="xs:boolean"/>

XML Schema Example

employee.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.abc.com"
xmlns="http://www.abc.com"
elementFormDefault="qualified">
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

employee.xml

```
<?xml version="1.0"?>
<employee
xmlns="http://www.abc.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.abc.com employee.xsd">
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

Description of XML Schema

<xs:element name="employee"> : It defines the element name employee.
<xs:complexType> : It defines that the element 'employee' is complex type.
<xs:sequence> : It defines that the complex type is a sequence of elements.
<xs:element name="firstname" type="xs:string"/> : It defines that the element 'firstname' is of string/text type.

XML Schema Data types

There are two types of data types in XML schema.

1. simpleType
2. complexType

simpleType

The simpleType allows you to have text-based elements. It contains less attributes, child elements, and cannot be left empty.

complexType

The complexType allows you to hold multiple attributes and elements. It can contain additional sub elements and can be left empty

DTD vs XSD

No.	DTD	XSD
1)	DTD stands for **Document Type Definition**.	XSD stands for XML Schema Definition.
2)	DTDs are derived from **SGML** syntax.	XSDs are written in XML.
3)	DTD **doesn't support datatypes**.	XSD **supports datatypes** for elements and attributes.
4)	DTD **doesn't support namespace**.	XSD **supports namespace**.
5)	DTD **doesn't define order** for child elements.	XSD **defines order** for child elements.
6)	DTD is **not extensible**.	XSD is **extensible**.
7)	DTD is **not simple to learn**.	XSD is **simple to learn** because you don't need to learn new language.
8)	DTD provides **less control** on XML structure.	XSD provides **more control** on XML structure.

CDATA

CDATA: (Unparsed Character data): CDATA contains the text which is not parsed further in an XML document. Tags inside the CDATA text are not treated as markup and entities will not be expanded.

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
<![CDATA[
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
]]>
</employee>
```

PCDATA

PCDATA: (Parsed Character Data): XML parsers are used to parse all the text in an XML document. PCDATA stands for Parsed Character data. PCDATA is the text that will be parsed by a parser. Tags inside the PCDATA will be treated as markup and entities will be expanded.

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

XML CSS

Purpose of CSS in XML

CSS (Cascading Style Sheets) can be used to add style and display information to an XML document. It can format the whole XML document.

How to link XML file with CSS

To link XML files with CSS, you should use the following syntax:

```
<?xml-stylesheet type="text/css" href="cssemployee.css"?>
```

XML CSS Example

cssemployee.css

```
employee
{
background-color: pink;
}
firstname,lastname,email
{
font-size:25px;
display:block;
color: blue;
margin-left: 50px;
}
```

employee.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="cssemployee.css"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

Example 1.

In this example, the XML file is created that contains the information about five books and displaying the XML file using CSS.

Books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="Rule.css"?>
<books>
        <heading>Welcome To Library </heading>
        <book>
                <title>Title -: Web Programming</title>
                <author>Author -: Chrisbates</author>
                <publisher>Publisher -: Wiley</publisher>
                <edition>Edition -: 3</edition>
                <price> Price -: 300</price>
        </book>
        <book>
                <title>Title -: Internet world-wide-web</title>
```

```
		<author>Author -: Ditel</author>
		<publisher>Publisher -: Pearson</publisher>
		<edition>Edition -: 3</edition>
		<price>Price -: 400</price>
	</book>
	<book>
		<title>Title -: Computer Networks</title>
		<author>Author -: Foruouzan</author>
		<publisher>Publisher -: Mc Graw Hill</publisher>
		<edition>Edition -: 5</edition>
		<price>Price -: 700</price>
	</book>
	<book>
		<title>Title -: DBMS Concepts</title>
		<author>Author -: Navath</author>
		<publisher>Publisher -: Oxford</publisher>
		<edition>Edition -: 5</edition>
		<price>Price -: 600</price>
	</book>
	<book>
		<title>Title -: Linux Programming</title>
		<author>Author -: Subhitab Das</author>
		<publisher>Publisher -: Oxford</publisher>
		<edition>Edition -: 8</edition>
		<price>Price -: 300</price>
	</book>
</books>
```

Rule.css

```
books {
	color: white;
	background-color : gray;
	width: 100%;
}
heading {
	color: green;
	font-size : 40px;
	background-color : powderblue;
}
heading, title, author, publisher, edition, price {
	display : block;
}
title {
	font-size : 25px;
	font-weight : bold;
}
```

in this example, the XML file is created that contains the information about various sections in Geeks for Geeks and the topics they contains and after that displaying the XML file using CSS .

Section.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="Geeks.css"?>
<Geeks_for_Geeks>
	<title>Hello Everyone! Welcome to Library</title>
	<geeks_section>
		<name>Algo</name>
		<topic1>Greedy Algo</topic1>
		<topic2>Randomised Algo</topic2>
		<topic3>Searching Algo</topic3>
		<topic4>Sorting Algo</topic4>
	</geeks_section>
	<geeks_section>
		<name>Data Structures</name>
		<topic1>Array</topic1>
		<topic2>Stack</topic2>
		<topic3>Queue</topic3>
		<topic4>Linked List</topic4>
	</geeks_section>
	<geeks_section>
		<name>Web Technology</name>
		<topic1>HTML</topic1>
		<topic2>CSS</topic2>
		<topic3>Java Script</topic3>
		<topic4>Php</topic4>
	</geeks_section>
	<geeks_section>
		<name>Languages</name>
		<topic1>C/C++</topic1>
		<topic2>Java</topic2>
		<topic3>Python</topic3>
		<topic4>Ruby</topic4>
	</geeks_section>
	<geeks_section>
		<name>DBMS</name>
		<topic1>Basics</topic1>
		<topic2>ER Diagram</topic2>
		<topic3>Normalization</topic3>
		<topic4>Transaction Concepts</topic4>
	</geeks_section>
</Geeks_for_Geeks>
```

Geeks.css

```
Geeks_for_Geeks
                        {
                        font-size:80%;
                        margin:0.5em;
                        font-family: Verdana;
                        display:block;
                        }
geeks_section {
                        display:block;
                        border: 1px solid silver;
                        margin:0.5em;
                        padding:0.5em;
                        background-color:whitesmoke;
                        }
title {
        display:block;
        font-weight:bolder;
        text-align:center;
        font-size:30px;
        background-color: green;
        color: white;

        }
name, topic1, topic2, topic3, topic4
{
display: block;
text-align: center;
}
name {
        color: green;
        text-decoration: underline ;
        font-weight: bolder;
        font-size:20px;
        }
topic1 {
        color: green
        }
topic2 {
        color: brown
        }
topic3 {
        color: blue
        }
```

```
topic4 {
        color: orange }
```

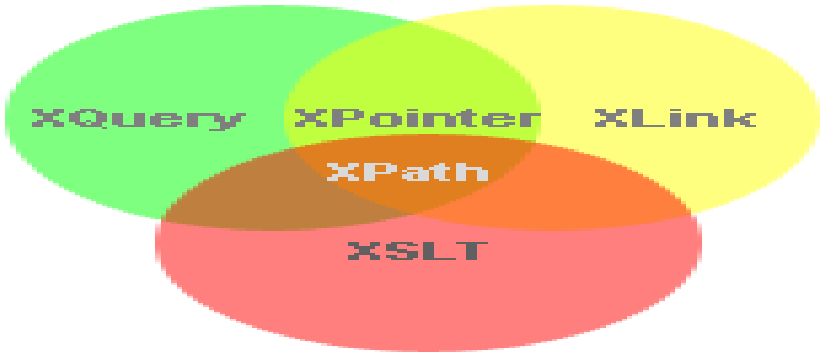
WHAT IS XPATH?

XPath is an official recommendation of the World Wide Web Consortium (W3C). It defines a language to find information in an XML file. It is used to traverse elements and attributes of an XML document. XPath provides various types of expressions which can be used to enquire relevant information from the XML document.

- **Structure Definitions** − XPath defines the parts of an XML document like element, attribute, text, namespace, processing-instruction, comment, and document nodes
- **Path Expressions** − XPath provides powerful path expressions select nodes or list of nodes in XML documents.
- **Standard Functions** − XPath provides a rich library of standard functions for manipulation of string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values etc.
- **Major part of XSLT** − XPath is one of the major elements in XSLT standard and is must have knowledge in order to work with XSLT documents.
- **W3C recommendation** − XPath is an official recommendation of World Wide Web Consortium (W3C).

One should keep the following points in mind, while working with XPath −

- XPath is core component of XSLT standard.
- XSLT cannot work without XPath.
- XPath is basis of XQuery and XPointer.



- Path stands for XML Path Language
- XPath uses "path like" syntax to identify and navigate nodes in an XML document
- XPath contains over 200 built-in functions

- XPath is a major element in the XSLT standard
- XPath is a W3C recommendation

Important features of XPath

- **XPath defines structure:** XPath is used to define the parts of an XML document i.e. element, attributes, text, namespace, processing-instruction, comment, and document nodes.
- **XPath provides path expression:** XPath provides powerful path expressions, select nodes, or list of nodes in XML documents.
- **XPath is a core component of XSLT:** XPath is a major element in XSLT standard and must be followed to work with XSLT documents.
- **XPath is a standard function:** XPath provides a rich library of standard functions to manipulate string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values etc.
- **Path is W3C recommendation.**

XPath Path Expressions

XPath uses path expressions to select nodes or node-sets in an XML document.

These path expressions look very much like the path expressions you use with traditional computer file systems:

XPath Standard Functions

XPath includes over 200 built-in functions.

There are functions for string values, numeric values, booleans, date and time comparison, node manipulation, sequence manipulation, and much more.

Today XPath expressions can also be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

XPath is Used in XSLT

XPath is a major element in the XSLT standard.

With XPath knowledge you will be able to take great advantage of your XSLT knowledge.

XPath is a W3C Recommendation

XPath 1.0 became a W3C Recommendation on November 16, 1999.

XPath 2.0 became a W3C Recommendation on January 23, 2007.

XPath 3.0 became a W3C Recommendation on April 8, 2014.

XPath Terminology

Nodes

In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document nodes.
XML documents are treated as trees of nodes. The topmost element of the tree is called the root element.
Look at the following XML document:

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

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Example of nodes in the XML document above:
<bookstore> (root element node)

<author>J K. Rowling</author> (element node)

lang="en" (attribute node)
Atomic values
Atomic values are nodes with no children or parent.
Example of atomic values:
J K. Rowling

"en"
Items
Items are atomic values or nodes.
Relationship of Nodes
Parent
Each element and attribute has one parent.
In the following example; the book element is the parent of the title, author, year, and price:
Children
Element nodes may have zero, one or more children.
In the following example; the title, author, year, and price elements are all children of the book element
Siblings
Nodes that have the same parent.
In the following example; the title, author, year, and price elements are all siblings:
Ancestors
A node's parent, parent's parent, etc.
In the following example; the ancestors of the title element are the book element and the bookstore element:
Descendants
A node's children, children's children, etc.

In the following example; descendants of the bookstore element are the book, title, author, year, and price elements:
XPath uses path expressions to select nodes or node-sets in an XML document. The node is selected by following a path or steps.
The XML Example Document
We will use the following XML document in the examples below.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>
<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>
</bookstore>
```

Selecting Nodes
XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
bookstore	Selects all nodes with the name "bookstore"
/bookstore	Selects the root element bookstore
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document

bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang

Predicates
Predicates are used to find a specific node or a node that contains a specific value.
Predicates are always embedded in square brackets.
In the table below we have listed some path expressions with predicates and the result of the expressions:

Path Expression	**Result**
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element. **Note:** In IE 5,6,7,8,9 first node is[0], but according to W3C, it is [1]. To solve this problem in IE, set the SelectionLanguage to XPath: *In JavaScript:* *xml*.setProperty("SelectionLanguage","XPath");
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

Selecting Unknown Nodes
XPath wildcards can be used to select unknown XML nodes.

Wildcard	**Description**

*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
/bookstore/*	Selects all the child element nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have at least one attribute of any kind

Selecting Several Paths
By using the | operator in an XPath expression you can select several paths.
In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
//book/title | //book/price	Selects all the title AND price elements of all book elements
//title | //price	Selects all the title AND price elements in the document
/bookstore/book/title | //price	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

The XML Example Document
"books.xml":

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

<bookstore>

<book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="children">
```

```
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="web">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="web">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

Loading the XML Document

Using an XMLHttpRequest object to load XML documents is supported in all modern browsers.

```
var xmlhttp = new XMLHttpRequest();
```

Code for older browsers (IE5 and IE6) can be found in the AJAX tutorial.

Selecting Nodes

Unfortunately, there are different ways of dealing with XPath in different browsers.
Chrome, Firefox, Edge, Opera, and Safari use the evaluate() method to select nodes:

```
xmlDoc.evaluate(xpath, xmlDoc, null, XPathResult.ANY_TYPE,null);
```

Internet Explorer uses the selectNodes() method to select node:

```
xmlDoc.selectNodes(xpath);
```

In our examples we have included code that should work with most major browsers.

Select all the titles

The following example selects all the title nodes:

Example

```
/bookstore/book/title
```

Select the title of the first book

The following example selects the title of the first book node under the bookstore element:

Example

```
/bookstore/book[1]/title
```

Select all the prices

The following example selects the text from all the price nodes:

Example

```
/bookstore/book/price[text()]
```

Select price nodes with price>35

The following example selects all the price nodes with a price higher than 35:

Example

```
/bookstore/book[price>35]/price
```

Select title nodes with price>35

The following example selects all the title nodes with a price higher than 35:

Example

```
/bookstore/book[price>35]/title
```

XSL

Before learning XSLT, we should first understand XSL which stands for E**X**tensible **S**tylesheet **L**anguage. It is similar to XML as CSS is to HTML.

Need for XSL

In case of HTML document, tags are predefined such as table, div, and span; and the browser knows how to add style to them and display those using CSS styles. But in case of XML documents, tags are not predefined. In order to understand and style an XML document, World Wide Web Consortium (W3C) developed XSL which can act as XML based Stylesheet Language. An XSL document specifies how a browser should render an XML document.

Following are the main parts of XSL −

- **XSLT** − used to transform XML document into various other types of document.
- **XPath** − used to navigate XML document.
- **XSL-FO** − used to format XML document.

What is XSLT

XSLT, Extensible Stylesheet Language Transformations, provides the ability to transform XML data from one format to another automatically.

CSS = Style Sheets for HTML

HTML uses predefined tags. The meaning of, and how to display each tag is well understood.

CSS is used to add styles to HTML elements.

XSL = Style Sheets for XML

XML does not use predefined tags, and therefore the meaning of each tag is not well understood.

A <table> element could indicate an HTML table, a piece of furniture, or something else - and browsers do not know how to display it!

So, XSL describes how the XML elements should be displayed.

XSLT = XSL Transformations

XSLT is the most important part of XSL.

XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.

With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

A common way to describe the transformation process is to say that **XSLT transforms an XML source-tree into an XML result-tree**.

XSLT Uses XPath

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

XSL - More Than a Style Sheet Language

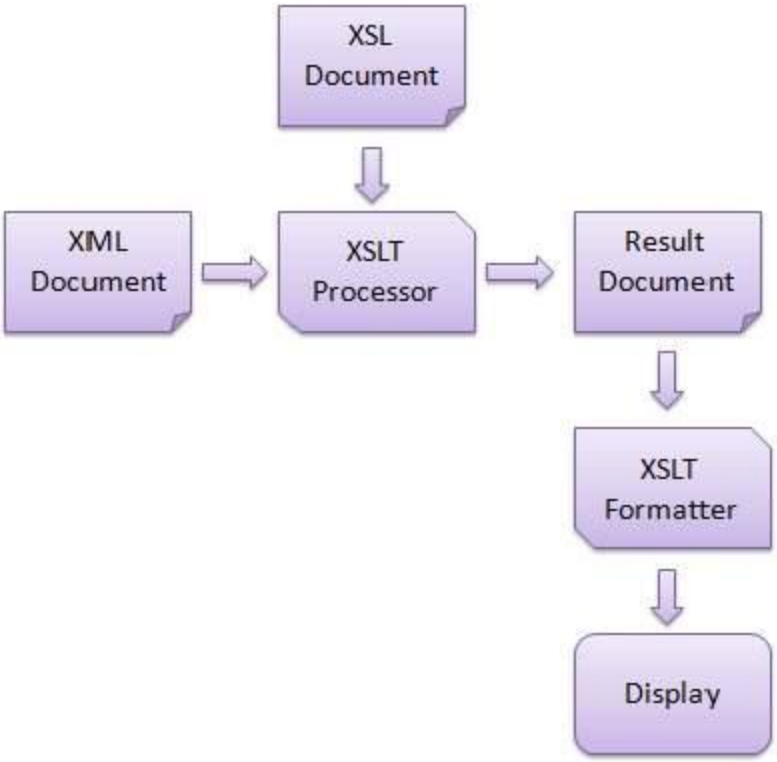
XSL consists of four parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents (discontinued in 2013)
- XQuery - a language for querying XML documents

How XSLT Works

An XSLT stylesheet is used to define the transformation rules to be applied on the target XML document. XSLT stylesheet is written in XML format. XSLT Processor takes the XSLT

stylesheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format. This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.



Advantages

Here are the advantages of using XSLT −

- Independent of programming. Transformations are written in a separate xsl file which is again an XML document.
- Output can be altered by simply modifying the transformations in xsl file. No need to change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.

Declaration

Following is the syntax declaration of **<xsl:template>** element.

```
<xsl:template
  name = Qname
  match = Pattern
  priority = number
  mode = QName >
</xsl:template>
```

Attributes

Sr.No	Name & Description
1	**name** Name of the element on which template is to be applied.
2	**match** Pattern which signifies the element(s) on which template is to be applied.
3	**priority** Priority number of a template. Matching template with low priority is not considered in from in front of high priority template.
4	**mode** Allows element to be processed multiple times to produce a different result each time.

- Elements

Number of occurrences	Unlimited
Parent elements	xsl:stylesheet, xsl:transform
Child elements	xsl:apply-imports,xsl:apply-templates,xsl:attribute, xsl:call-template, xsl:choose, xsl:comment, xsl:copy, xsl:copy-of, xsl:element, xsl:fallback, xsl:for-each, xsl:if, xsl:message, xsl:number, xsl:param, xsl:processing-instruction, xsl:text, xsl:value-of, xsl:variable, output elements

XML Parsers

An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML.

XML parser validates the document and check that the document is well formatted.
Let's understand the working of XML parser by the figure given below:

Types of XML Parsers

These are the two main types of XML Parsers:

Types of XML Parsers
These are the two main types of XML Parsers:

1. DOM
2. SAX

DOM (Document Object Model)

A DOM document is an object which contains all the information of an XML document. It is composed like a tree structure. The DOM Parser implements a DOM API. This API is very simple to use.

Features of DOM Parser

A DOM Parser creates an internal structure in memory which is a DOM document object and the client applications get information of the original XML document by invoking methods on this document object.
DOM Parser has a tree based structure.

Advantages

1) It supports both read and write operations and the API is very simple to use.
2) It is preferred when random access to widely separated parts of a document is required.

Disadvantages

1) It is memory inefficient. (consumes more memory because the whole XML document needs to loaded into memory).
2) It is comparatively slower than other parsers.

SAX (Simple API for XML)

A SAX Parser implements SAX API. This API is an event based API and less intuitive.

Features of SAX Parser

It does not create any internal structure.
Clients does not know what methods to call, they just overrides the methods of the API and place his own code inside method.
It is an event based parser, it works like an event handler in Java.

Advantages

1) It is simple and memory efficient.
2) It is very fast and works for huge documents.

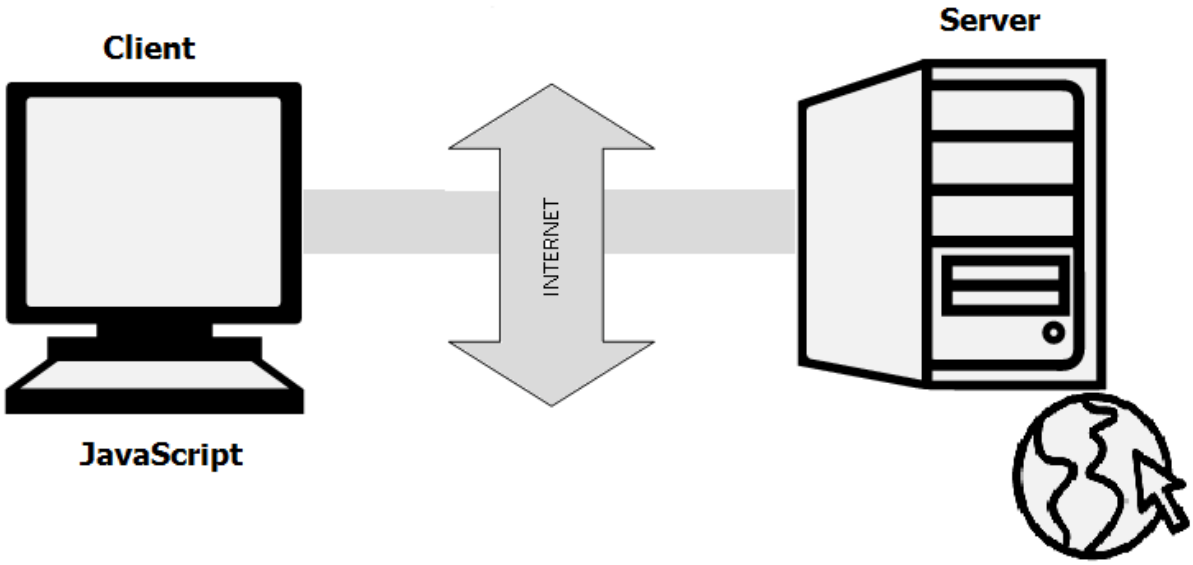
Disadvantages

1) It is event-based so its API is less intuitive.
2) Clients never know the full information because the data is broken into pieces.

UNIT III

What is JavaScript?

JavaScript is a very powerful client-side scripting language. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and Mobile application development.



You should place all your JavaScript code within **<script> tags**(<script> and </script>) if you are keeping your JavaScript code within the HTML document itself. You have to use the type attribute within the <script> tag and set its value to text/javascript like this:

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

Hello World Example:

```
<html>
<head>
      <title>My First JavaScript code!!!</title>
      <script type="text/javascript">
            alert("Hello World!");
      </script>
</head>
```

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

JavaScript Variables

Variables are used to **store values** (name = "John") **or expressions** (sum = x + y).

Declaration of variable:

```
var name;
```

Assigning value to variable:

```
var name;
name = "John";
```

One Statement, Many Variables:

You can declare many variables in one statement.

Start the statement with var and separate the variables by **comma**:

```
var person = "John Doe", carName = "Volvo", price = 200;
```

A declaration can span multiple lines:

```
var person="John Doe",
carName="Volvo",
price=200;
```

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

Example

```
var carName = "Volvo";
var carName;
```

Naming Variables: Though you can name the variables as you like, it is a good programming practice to give descriptive and meaningful names to the variables. Moreover, variable names should start with a letter and they are case sensitive. Hence the variables student name and studentName are different because the letter n in a name is different (n and N).

Consider following example for variables:-

```
<html>
<head>
<title>Variables!!!</title>
<script type="text/javascript">
var one = 22;
var two = 3;
var add = one + two;
var minus = one - two;
var multiply = one * two;
var divide = one/two;
      document.write("First No: = " + one + "<br />Second No: = " + two
+ " <br />");
      document.write(one + " + " + two + " = " + add + "<br/>");
      document.write(one + " - " + two + " = " + minus + "<br/>");
      document.write(one + " * " + two + " = " + multiply + "<br/>");
      document.write(one + " / " + two + " = " + divide + "<br/>");
</script>
</head>
<body>
</body>
</html>
```

JAVASCRIPT FUNCTIONS

What is Function in JavaScript?

Functions are very important and useful in any programming language as they make the code reusable. A function is a block of code which will be executed only if it is called. If you have a few lines of code that needs to be used several times, you can create a function including the repeating lines of code and then call the function wherever you want.

How to Create a Function in JavaScript

Use the keyword **function** followed by the name of the function.

After the function name, open and close parentheses.

After parenthesis, open and close curly braces.

Within curly braces, write your lines of code.

Syntax:

```
function functionname() {
     lines of code to be executed
}
```

Consider the following example:-

```
<html>
<head>
      <title>Functions!!!</title>
      <script type="text/javascript">
       function myFunction()
       {
            document.write("This is a simple function.<br />");
       }
            myFunction();
      </script>
</head>
<body>
</body>
</html>
```

Function with Arguments

You can create functions with arguments as well. Arguments should be specified within parenthesis

Syntax:

```
function functionname(arg1, arg2)
{
  lines of code to be executed
}
```

Consider the following example:-

```
<html>
<head>
      <script type="text/javascript">
            var count = 0;
            function countVowels(name)
            {
                  for (var i=0;i<name.length;i++)
                  {
               if(name[i] == "a" || name[i] == "e" || name[i] == "i" ||
name[i] == "o" || name[i] == "u")
               count = count + 1;
                  }
            document.write("Hello " + name + "!!! Your name has " +
count + " vowels.");
            }
            var myName = prompt("Please enter your name");
      countVowels(myName);
      </script>
</head>
<body>
</body>
</html>
```

JavaScript Return Value

You can also create JS functions that return values. Inside the function, you need to use the keyword **return** followed by the value to be returned.

Syntax:

```
function functionname(arg1, arg2)
{
  lines of code to be executed
  return val1;
}
```

Example:-

```
<html>
<head>
      <script type="text/javascript">
            function returnSum(first, second)
        {
          var sum = first + second;
          return sum;
```

```
        }
      var firstNo = 78;
      var secondNo = 22;
      document.write(firstNo + " + " + secondNo + " = " +
returnSum(firstNo,secondNo));
     </script>
</head>
<body>
</body>
</html>
```

JavaScript Operators

The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator. This is different from algebra. The following does not make sense in algebra:

x = x + 5

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

The "equal to" operator is written like == in JavaScript.

The **assignment** operator (=) assigns a value to a variable.

Assignment:-

```
var x = 10;
```

The **addition** operator (+) adds numbers:

Adding

```
var x = 5;
var y = 2;
var z = x + y;
```

The **multiplication** operator (*) multiplies numbers.

Multiplying

```
var x = 5;
var y = 2;
var z = x * y;
```

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus
++	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

JavaScript String Operators

The + operator can also be used to add (concatenate) strings.

Example

```
var txt1 = "John";
var txt2 = "Doe";
var txt3 = txt1 + " " + txt2;
```

The result of txt3 will be:

John Doe

The += assignment operator can also be used to add (concatenate) strings:

Example

```
var txt1 = "What a very ";
txt1 += "nice day";
```

The result of txt1 will be:

What a very nice day

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;
```

The result of *x*, *y*, and *z* will be:

10
55
Hello5

JavaScript Logical Operators

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

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 Type Operators

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

Operators and Operands

The numbers (in an arithmetic operation) are called **operands**.

The operation (to be performed between the two operands) is defined by an **operator**.

Operand	Operator	Operand
100	+	50

Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

Example

```
var x = 100 + 50 * 3;
```

Is the result of example above the same as 150 * 3, or is it the same as 100 + 150?

Is the addition or the multiplication done first?

As in traditional school mathematics, the multiplication is done first.

Multiplication (*) and division (/) have higher **precedence** than addition (+) and subtraction (-).

And (as in school mathematics) the precedence can be changed by using parentheses:

Example

```
var x = (100 + 50) * 3;
```

When using parentheses, the operations inside the parentheses are computed first.

When many operations have the same precedence (like addition and subtraction), they are computed from left to right:

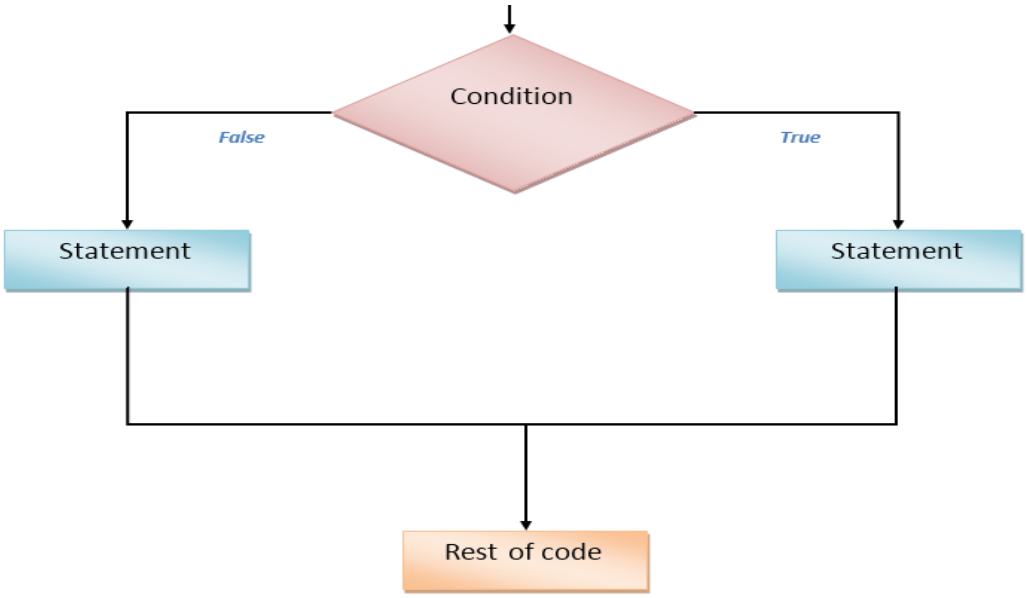
Example

```
var x = 100 + 50 - 3;
```

JavaScript Conditional Statements: IF, Else, Else IF

How to use Conditional Statements

Conditional statements are used to decide the flow of execution based on different conditions. If a condition is true, you can perform one action and if the condition is false, you can perform another action.



Different Types of Conditional Statements

There are mainly three types of conditional statements in JavaScript.

If statement

If…Else statement

If…Else If…Else statement

If statement

Syntax:

if (condition)

{

lines of code to be executed if condition is true

}

You can use If statement if you want to check only a specific condition.

Example:-

```
<html>
<head>
      <title>IF Statments!!!</title>
      <script type="text/javascript">
            var age = prompt("Please enter your age");
            if(age>=18)
            document.write("You are an adult <br />");
            if(age<18)
            document.write("You are NOT an adult <br />");
      </script>
</head>
<body>
</body>
</html>
```

If…Else statement

Syntax:

```
if (condition)
{
lines of code to be executed if the condition is true
}
else
{
lines of code to be executed if the condition is false
}
```

You can use If….Else statement if you have to check two conditions and execute a different set of codes.

Example:-

```
<html>
```

```
<head>
      <title>If...Else Statments!!!</title>
      <script type="text/javascript">
            // Get the current hours
            var hours = new Date().getHours();
            if(hours<12)
            document.write("Good Morning!!!<br />");
            else
            document.write("Good Afternoon!!!<br />");
      </script>
</head>
<body>
</body>
</html>
```

If…Else If…Else statement

Syntax:

```
if (condition1)
{
      lines of code to be executed if condition1 is true
}
else if(condition2)
{
      lines of code to be executed if condition2 is true
}
else
{
      lines of code to be executed if condition1 is false and
condition2 is false
}
```

You can use If….Else If….Else statement if you want to check more than two conditions.

Example:-

```
<html>
<head>
      <script type="text/javascript">
            var one = prompt("Enter the first number");
            var two = prompt("Enter the second number");
            one = parseInt(one);
            two = parseInt(two);
            if (one == two)
```

```
                document.write(one + " is equal to " + two + ".");
          else if (one<two)
                document.write(one + " is less than " + two + ".");
          else
                document.write(one + " is greater than " + two + ".");
     </script>
</head>
<body>
</body>
</html>
```

JavaScript Switch Statement

The switch statement is used to perform different actions based on different conditions.

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

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

The switch expression is evaluated once.

The value of the expression is compared with the values of each case.

If there is a match, the associated block of code is executed.

Example

The getDay() method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

The result of day will be:

Monday

JavaScript For Loop

Loops can execute a block of code a number of times. 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:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
var i;
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

Different Kinds of Loops

JavaScript supports different kinds of loops:

`for` - loops through a block of code a number of times

`for/in` - loops through the properties of an object

`while` - loops through a block of code while a specified condition is true

`do/while` - also loops through a block of code while a specified condition is true

The For Loop

The `for` loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

Example

```
for (i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

The For/In Loop

The JavaScript `for/in` statement loops through the properties of an object:

Example

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
  text += person[x];
}
```

The While Loop

The while loop loops through a block of code as long as a specified condition is true.

Syntax

```
while (condition) {
  // code block to be executed
}
```

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
  // code block to be executed
}
while (condition);
```

Example

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

JavaScript Break and Continue

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

The Break Statement

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.

The break statement can also be used to jump out of a loop.

The break statement breaks the loop and continues executing the code after the loop (if any):

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

The Continue Statement

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

JavaScript Arrays

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

Example

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

What is an Array?

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.

Creating an Array

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

Syntax:

```
var array_name = [item1, item2, ...];
```

Example

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

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

```
var cars = [
  "Saab",
  "Volvo",
```

```
  "BMW"
];
```

Access the Elements of an Array

You access an array element by referring to the **index number**.

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

```
var name = cars[0];
```

Example

```
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:

```
cars[0] = "Opel";
```

Example

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

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

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

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

Examples

```
var x = cars.length;   // The length property returns the number of
elements
var y = cars.sort();   // The sort() method sorts arrays
```

Accessing the First Array Element

Example

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
```

Accessing the Last Array Element

Example

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var last = fruits[fruits.length - 1];
```

Adding Array Elements

The easiest way to add a new element to an array is using the push() method:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");    // adds a new element (Lemon) to fruits
```

New element can also be added to an array using the length property:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";    // adds a new element (Lemon) to
fruits
```

JavaScript Array Methods

Converting Arrays to Strings

The JavaScript method toString() converts an array to a string of (comma separated) array values.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result:

Banana,Orange,Apple,Mango

The join() method also joins all array elements into a string.

It behaves just like toString(), but in addition you can specify the separator:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Result:

Banana * Orange * Apple * Mango

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

Popping

The pop() method removes the last element from an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();              // Removes the last element ("Mango") from
fruits
```

The pop() method returns the value that was "popped out":

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.pop();      // the value of x is "Mango"
```

Pushing

The push() method adds a new element to an array (at the end):

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");       //  Adds a new element ("Kiwi") to fruits
```

The push() method returns the new array length:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");   //  the value of x is 5
```

JavaScript Strings

Method	Description
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a new joined strings
endsWith()	Checks whether a string ends with specified string/characters
fromCharCode()	Converts Unicode values to characters
includes()	Checks whether a string contains the specified string/characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string

localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches
repeat()	Returns a new string with a specified number of copies of an existing string
replace()	Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced
search()	Searches a string for a specified value, or regular expression, and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
startsWith()	Checks whether a string begins with specified characters
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale

toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

String Properties

Property	Description
constructor	Returns the string's constructor function

length	Returns the length of a string
prototype	Allows you to add properties and methods to an object

String Methods

All string methods return a new value. They do not change the original variable.

String Length

The length property returns the length of a string:

Example

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

Finding a String in a String

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

JavaScript counts positions from zero.
0 is the first position in a string, 1 is the second, 2 is the third ...

The lastIndexOf() method returns the index of the **last** occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Both indexOf(), and lastIndexOf() return -1 if the text is not found.

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("John");
```

Both methods accept a second parameter as the starting position for the search:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate", 15);
```

The lastIndexOf() methods searches backwards, meaning: if the second parameter is 15, the search starts at position 15, counting from the end, and searches to the beginning of the string.

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate", 15);
```

Searching for a String in a String

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

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

Extracting String Parts

There are 3 methods for extracting a part of a string:

slice(*start, end*)

substring(*start, end*)

substr(*start, length*)

The slice() Method

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

The result of res will be:

Banana

The substring() Method

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13);
```

The result of *res* will be:

Banana

The substr() Method

substr() is similar to slice().

The difference is that the second parameter specifies the **length** of the extracted part.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7, 6);
```

The result of res will be:

Banana

Replacing String Content

The replace() method replaces a specified value with another value in a string:

Example

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

Converting to Upper and Lower Case

A string is converted to upper case with toUpperCase():

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper
```

The concat() Method

concat() joins two or more strings:

Example

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
```

String.trim()

The trim() method removes whitespace from both sides of a string:

Example

```
var str = "       Hello World!        ";
alert(str.trim());
```

Extracting String Characters

There are 3 methods for extracting string characters:

charAt(*position*)

charCodeAt(*position*)

Property access [ ]

The charAt() Method

The charAt() method returns the character at a specified index (position) in a string:

Example

```
var str = "HELLO WORLD";
str.charAt(0);            // returns H
```

The charCodeAt() Method

The charCodeAt() method returns the unicode of the character at a specified index in a string:

The method returns a UTF-16 code (an integer between 0 and 65535).

Example

```
var str = "HELLO WORLD";
str.charCodeAt(0);          // returns 72
```

Property Access

ECMAScript 5 (2009) allows property access [ ] on strings:

Example

```
var str = "HELLO WORLD";
str[0];                     // returns H
```

Converting a String to an Array

A string can be converted to an array with the split() method:

Example

```
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split on commas
txt.split(" ");          // Split on spaces
txt.split("|");          // Split on pip
```

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

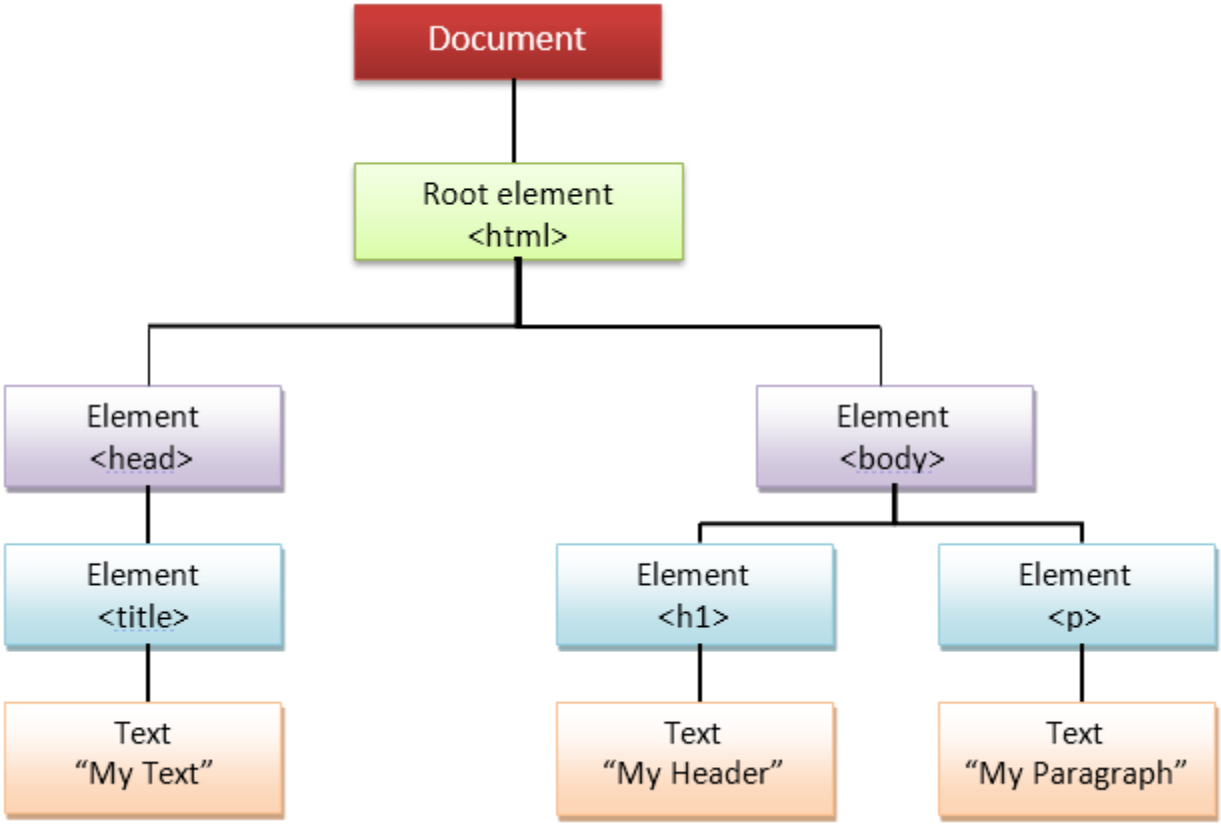
Core DOM - standard model for all document types

XML DOM - standard model for XML documents

HTML DOM - standard model for HTML documents

What is DOM in JavaScript?

JavaScript can access all the elements in a webpage making use of Document Object Model (DOM). In fact, the web browser creates a DOM of the webpage when the page is loaded. The DOM model is created as a tree of objects like this:



How to use DOM and Events

Using DOM, JavaScript can perform multiple tasks. It can create new elements and attributes, change the existing elements and attributes and even remove existing elements and attributes. JavaScript can also react to existing events and create new events in the page.

getElementById, innerHTML Example

getElementById: To access elements and attributes whose id is set.

innerHTML: To access the content of an element.

Try this Example yourself:

Top of Form

```
<html>
```

```
<head>
      <title>DOM!!!</title>
</head>
<body>
  <h1 id="one">Welcome</h1>
  <p>This is the welcome message.</p>
  <h2>Technology</h2>
  <p>This is the technology section.</p>
  <script type="text/javascript">
           var text = document.getElementById("one").innerHTML;
           alert("The first heading is " + text);
  </script>
</body>
</html>
```

getElementsByTagName Example

getElementsByTagName: To access elements and attributes using tag name. This method will return an array of all the items with the same tag name.

Try this Example yourself:

```
<html>
<head>
      <title>DOM!!!</title>
</head>
<body>
  <h1>Welcome</h1>
  <p>This is the welcome message.</p>
  <h2>Technology</h2>
  <p id="second">This is the technology section.</p>
  <script type="text/javascript">
      var paragraphs = document.getElementsByTagName("p");
    alert("Content in the second paragraph is " +
paragraphs[1].innerHTML);
    document.getElementById("second").innerHTML = "The orginal message
is changed.";
  </script>
</body>
</html>
```

Event handler Example

createElement: To create new element

removeChild: Remove an element

You can add an **event handler** to a particular element like this:

```
document.getElementById(id).onclick=function()
   {
      lines of code to be executed
   }
```

OR

```
document.getElementById(id).addEventListener("click", functionname)
```

Try this Example yourself:

```
<html>
<head>
     <title>DOM!!!</title>
</head>
<body>
  <input type="button" id="btnClick" value="Click Me!!" />
  <script type="text/javascript">
     document.getElementById("btnClick").addEventListener("click",
clicked);
    function clicked()
    {
            alert("You clicked me!!!");
    }
  </script>
</body>
</html>
```

JavaScript Closures

JavaScript variables can belong to the **local** or **global** scope.

Global variables can be made local (private) with **closures**.

Global Variables

A function can access all variables defined **inside** the function, like this:

Example

```
function myFunction() {
  var a = 4;
  return a * a;
}
```

But a function can also access variables defined **outside** the function, like this:

Example

```
var a = 4;
function myFunction() {
  return a * a;
}
```

In the last example, **a** is a **global** variable.

In a web page, global variables belong to the window object.

Global variables can be used (and changed) by all scripts in the page (and in the window).

In the first example, **a** is a **local** variable.

A local variable can only be used inside the function where it is defined. It is hidden from other functions and other scripting code.

Global and local variables with the same name are different variables. Modifying one, does not modify the other.

Variables created **without** the keyword **var**, are always global, even if they are created inside a function.

Variable Lifetime

Global variables live as long as your application (your window / your web page) lives.

Local variables have short lives. They are created when the function is invoked, and deleted when the function is finished.

JavaScript Closures

Remember self-invoking functions? What does this function do?

Example

```
var add = (function () {
  var counter = 0;
  return function () {counter += 1; return counter}
```

```
})();

add();
add();
add();

// the counter is now 3
```

Example Explained

The variable add is assigned the return value of a self-invoking function.

The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.

This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.

This is called a JavaScript **closure.** It makes it possible for a function to have "**private**" variables.

The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

A closure is a function having access to the parent scope, even after the parent function has closed.

AJAX:

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML.

Through AJAX you can:

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background
- Create better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.

- Data-driven as opposed to page-driven.

  Ex:Google Maps, Google Suggest ,Gmail, Yahoo Maps (new)

Rich Internet Application Technology:

AJAX is the most viable Rich Internet Application (RIA) technology so far. It is getting tremendous industry momentum and several tool kit and frameworks are emerging. But at the same time, AJAX has browser incompatibility and it is supported by JavaScript, which is hard to maintain and debug.

AJAX is Based on Open Standards:

AJAX is based on the following open standards –

Browser-based presentation using HTML and Cascading Style Sheets (CSS).

Data is stored in XML format and fetched from the server.

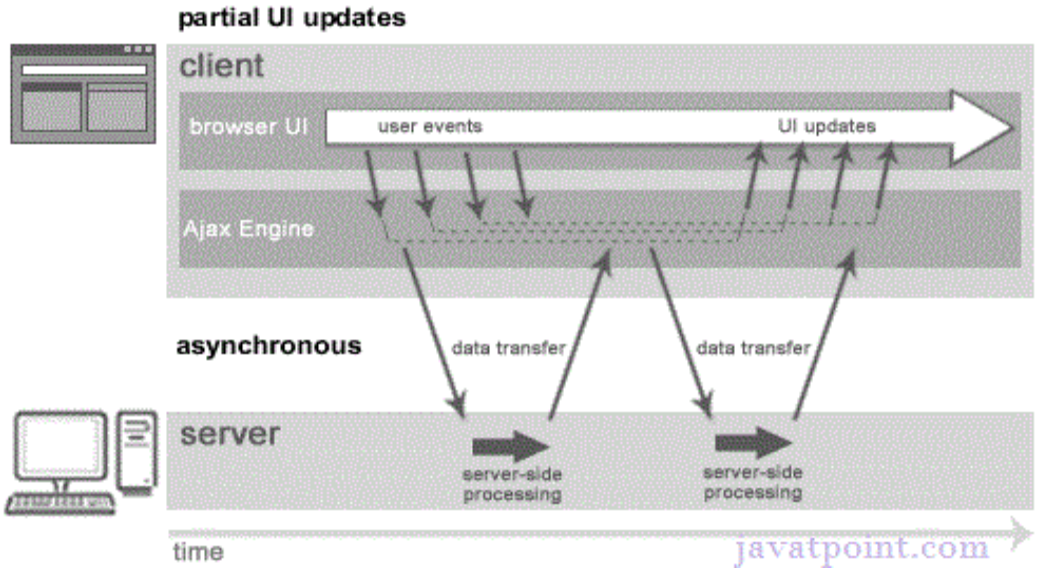
Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.

JavaScript to make everything happen.

There are too many web applications running on the web that are using ajax technology like **gmail**, **facebook**,**twitter**, **google map**, **youtube** etc.

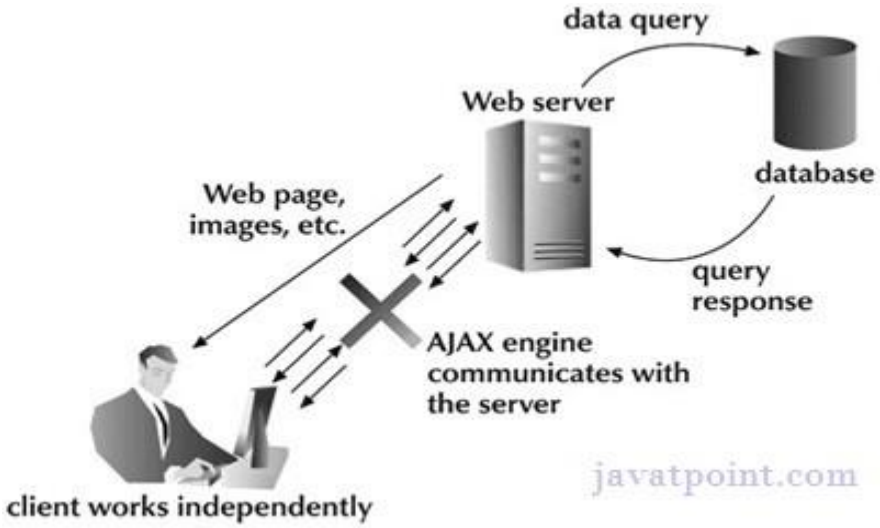
Asynchronous (AJAX Web-Application Model):

An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.



As you can see in the above image, full page is not refreshed at request time and user gets response from the ajax engine.

Let's try to understand asynchronous communication by the image given below.



Ajax Communication Techniques:

AJAX Technologies:

As describe earlier, ajax is not a technology but group of inter-related technologies. AJAX technologies includes:

HTML/XHTML and CSS

DOM

XML or JSON

XMLHttpRequest

JavaScript

HTML/XHTML and CSS:
These technologies are used for displaying content and style. It is mainly used for presentation.

DOM:
It is used for dynamic display and interaction with data.

XML or JSON:
For carrying data to and from server. JSON (Javascript Object Notation) is like XML but short and faster than XML.

XMLHttpRequest:
For asynchronous communication between client and server. For more visit next page.

JavaScript:
It is used to bring above technologies together.

Independently, it is used mainly for client-side validation.

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();

Example

var xhttp = new XMLHttpRequest();

The XMLHttpRequest object is the key to AJAX. It has been available ever since Internet Explorer 5.5 was released in July 2000, but was not fully discovered until AJAX and Web 2.0 in 2005 became popular.

XMLHttpRequest (XHR) is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side.

The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.

You already have seen a couple of examples on how to create an XMLHttpRequest object.

Listed below are some of the methods and properties that you have to get familiar with.

XMLHttpRequest Methods:

abort()

Cancels the current request.

getAllResponseHeaders()

Returns the complete set of HTTP headers as a string.

getResponseHeader(headerName)

Returns the value of the specified HTTP header.

open(method, URL)

open(method, URL, async)

open(method, URL, async, userName)

open(method, URL, async, userName, password)

Specifies the method, URL, and other optional attributes of a request.

The method parameter can have a value of "GET", "POST", or "HEAD". Other HTTP methods such as "PUT" and "DELETE" (primarily used in REST applications) may be possible.

The "async" parameter specifies whether the request should be handled asynchronously or not. "true" means that the script processing carries on after the send() method without waiting for a response, and "false" means that the script waits for a response before continuing script processing.

send(content)

Sends the request.

setRequestHeader(label, value)

Adds a label/value pair to the HTTP header to be sent.

XMLHttpRequest Properties:

onreadystatechange

An event handler for an event that fires at every state change.

readyState

The readyState property defines the current state of the XMLHttpRequest object.

The following table provides a list of the possible values for the readyState property −

State	Description
0	The request is not initialized.
1	The request has been set up.
2	The request has been sent.
3	The request is in process.
4	The request is completed.

readyState = 0 After you have created the XMLHttpRequest object, but before you have called the open() method.

readyState = 1 After you have called the open() method, but before you have called send().

readyState = 2 After you have called send().

readyState = 3 After the browser has established a communication with the server, but before the server has completed the response.

readyState = 4 After the request has been completed, and the response data has been completely received from the server.

responseText

Returns the response as a string.

responseXML

Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.

status

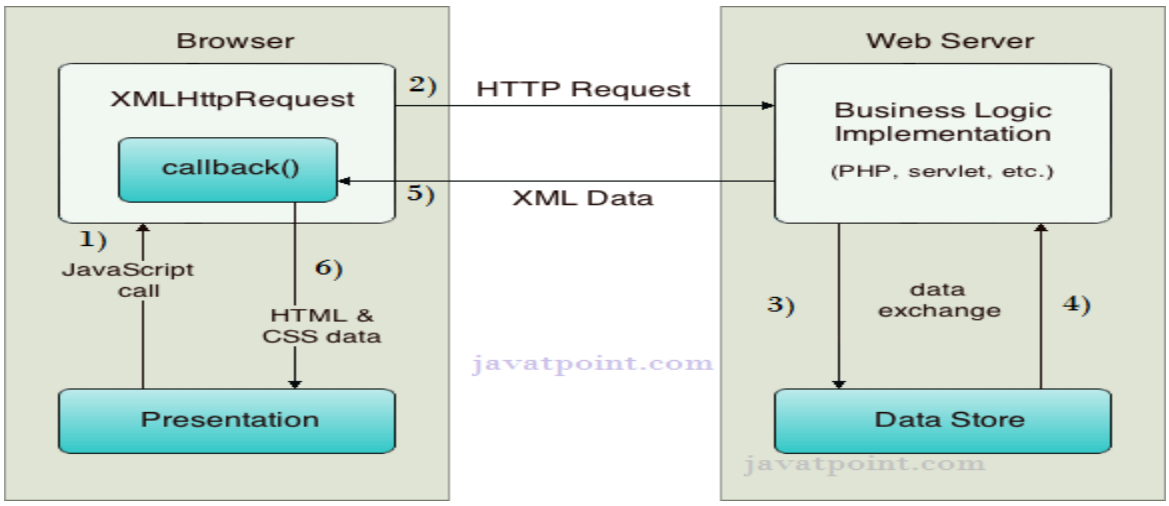
Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").

statusText

Returns the status as a string (e.g., "Not Found" or "OK").

How AJAX works?:

AJAX communicates with the server using XMLHttpRequest object. Let's try to understand the flow of ajax or how ajax works by the image displayed below.



As you can see in the above example, XMLHttpRequest object plays a important role.

User sends a request from the UI and a javascript call goes to XMLHttpRequest object.

HTTP Request is sent to the server by XMLHttpRequest object.

Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.

Data is retrieved.

Server sends XML data or JSON data to the XMLHttpRequest callback function.

HTML and CSS data is displayed on the browser.

AJAX data formats:

Upon sending an XMLHttpRequest request to the server, we will have a response. This response could be in two different formats: plain text or in XML.

- Plain text: we could receive just a text, a word, a sentence (JSON is actually a sentence)

- XML: the response we received will be XML encoded, so our JavaScript may need to perform some transformation to display it.

XML:

The eXtensible Markup Language was created to provide some structure to the information exchanged between multiple systems. It provides a way to store this information and to transport it.

Basically with XML you could create your own language to store your information, for instance:

?

```
1    <record>
2       <from>John Doe</from>
3       <to>Everyone</to>
4       <title>First XML Record</title>
5       <description>This is the first of many XML records</description>
6    </record>
```

To access an XML response, the XMLHttpRequest object has a property called: responseXML.

Once you have the response in a JS variable, you can access the different nodes of the XML with standard DOM functions: i.e: getElementsByTagName.

String:

As mentioned before, an XMLHttpRequest could receive plain text as a response: a word, a letter, a complete sence, and text with special encoding. To access this response you have to use a property called: responseText.

JSON:

JSON stands for JavaScript Object Notation which is a data interchange format. It is now widely adopted as a way to handle responses from server, since it is a simple encoding that allows human and machines to read it easily.

In our tutorial, as I briefly mentioned before, a JSON response is actually a text response, so you should use the responseText property to access it. Once you have it available, you could use multiple JSON libraries to parse it, and perform any operation needed based on the information received.

AJAX SECURITY CONCERNS:

The Ajax calls are sent in plain text format, this might lead to insecure database access. The data gets stored on the clients browser, thus making the data available to anyone. It also allows monitoring browsing sessions by inserting scripts.

AJAX function calls are sent in plain text to server. These calls may easily reveal database details, variable names etc

User’s browsing session can be monitored my maliciously inserting scripts

Ajax may encourage developers to use multiple server side pages thereby introducing multiple entry points for attackers

- A JavaScript can not access the local file system without the user's permission.
- An AJAX interaction can only be made with the servers-side component from which the page was loaded.
- A proxy pattern could be used for AJAX interactions with external services.
- The application model should not be exposed as some user might be able to reverse engineer the application.
- HTTPS can be used to secure the connection when confidential information is being exchanged

AJAX Security: Server Side:

AJAX-based Web applications use the same server-side security schemes of regular Web applications.

You specify authentication, authorization, and data protection requirements in your web.xml file (declarative) or in your program (programmatic).

AJAX-based Web applications are subject to the same security threats as regular Web applications.

AJAX Security: Client Side:

JavaScript code is visible to a user/hacker. Hacker can use JavaScript code for inferring server-side weaknesses.

JavaScript code is downloaded from the server and executed ("eval") at the client and can compromise the client by mal-intended code.

Downloaded JavaScript code is constrained by the sand-box security model and can be relaxed for signed JavaScript.

USER INTERFACE DESIGN FOR AJAX:

The user interface for ajax is designed using html,xml,javascript,css

File directory:

ajax_example.html

surprise.html

css (folder) containing styles.css file

you'll have a main HTML file, CSS stylesheet, JavaScript file, and an **additional** HTML file.

We're going to grab the contents of the **second** HTML file and insert it into the first page.

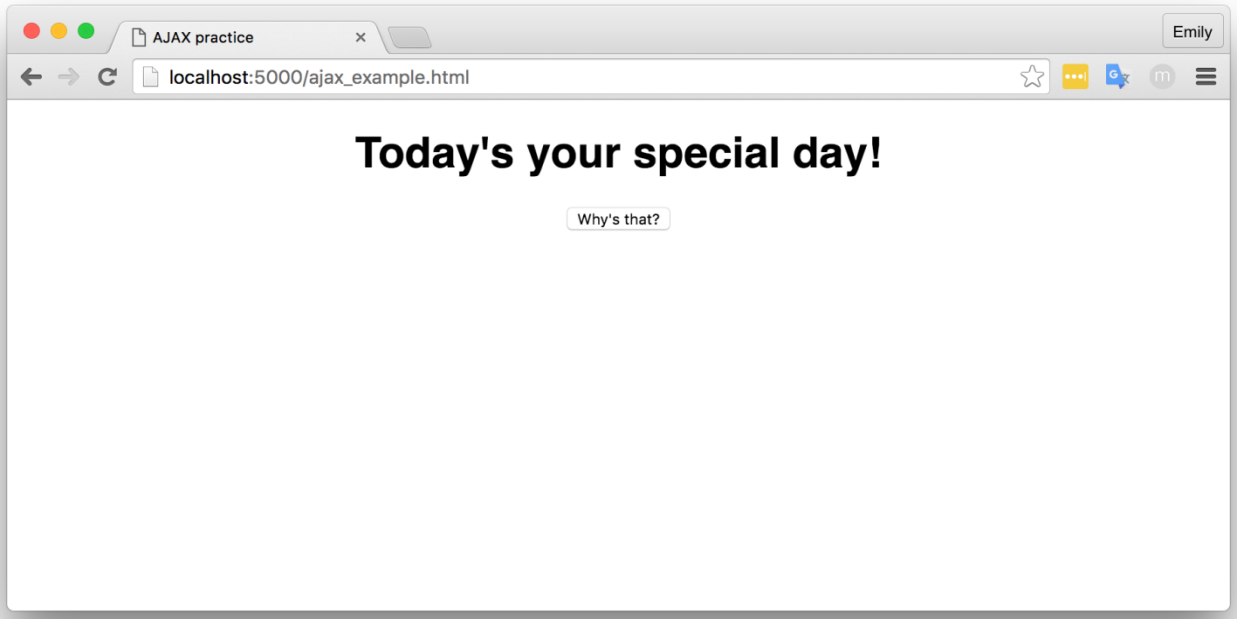
Here's the main HTML page.

```
<!DOCTYPE html>
<html>    <head>
        <meta charset="utf-8">
        <link rel="stylesheet" href="css/styles.css">
        <title>AJAX practice</title>
        <script>
        </script>
    </head>
    <body>
        <h1>Today's your special day!</h1>
        <button id="reveal">Why's that?</button>
        <div id="ajax-content">
        </div>
    </body>
</html>
```

Here's the content you'll put in surprise.html (if on your computer)

```
<h1 id="birthday-greeting">It's your birthday!</h1>
```

surprise.html is the file we'll load from within ajax_example.html - via AJAX!



Main HTML page

Introduction to XMLHttpRequest:

XMLHttpRequest is a mouthful. It's a system that lets data be transferred between a client and a server. As you learned, this normally happens via request and response. The same is true with XMLHttpRequest, except you can grab data from a URL without the page refreshing!

Think to a time you've used Facebook or Gmail. You've performed actions without reloading an entire page. You've left comments that post instantly while you're on the same page, for example. That's what AJAX allows!

Your first AJAX call:

1. First, you'll create an XMLHttpRequest object.
2. Open your request with the open method.
3. Send the request with the send method.

You need to create an XMLHttpRequest object for every AJAX request you make. Let's see how this looks in a code example.

```
// 1. create a new XMLHttpRequest object -- an object like any other!

var myRequest = new XMLHttpRequest();

// 2. open the request and pass the HTTP method name and the resource as parameters

myRequest.open('GET', 'surprise.html');

// 3. write a function that runs anytime the state of the AJAX request changes

myRequest.onreadystatechange = function () {

    // 4. check if the request has a readyState of 4, which indicates the server has responded (complete)

    if (myRequest.readyState === 4) {

        // 5. insert the text sent by the server into the HTML of the 'ajax-content'

        document.getElementById('ajax-content').innerHTML = myRequest.responseText;

    }

}
```

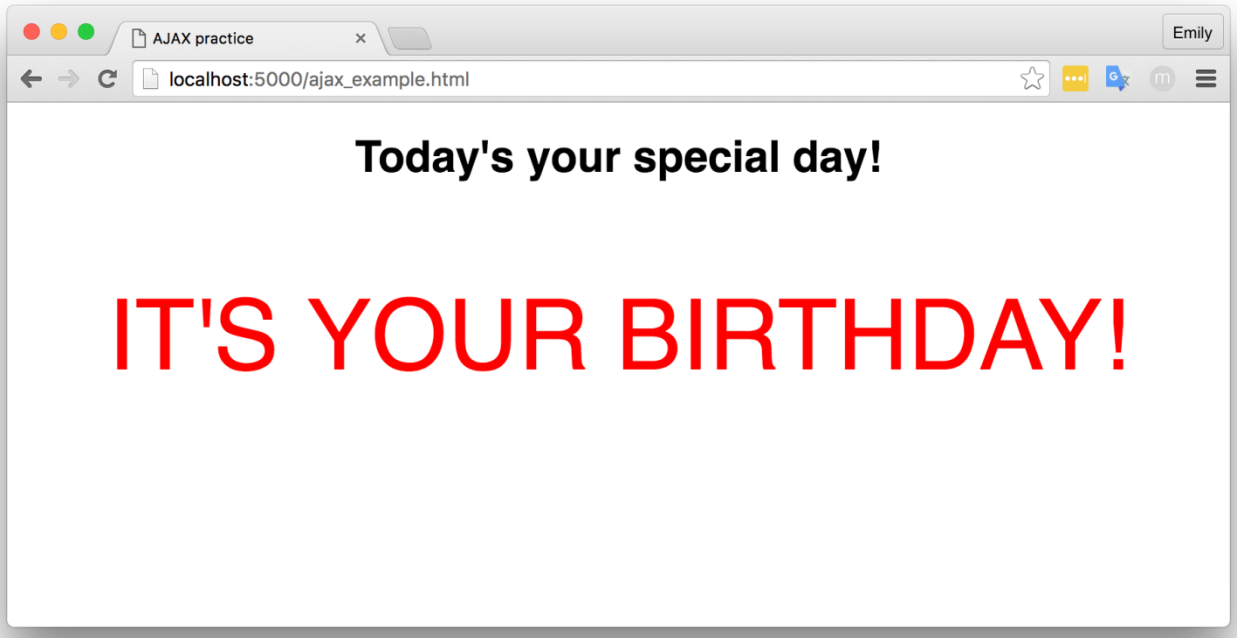
readyState can have a value between 0 and 4. You'll probably never use anything besides areadyState of 4, which indicates the server has sent back its full response.

Now, we need a function to call within the page when the button is clicked. Since a button click will send the AJAX, why not name it sendTheAJAX?

```
function sendTheAJAX() {

    myRequest.send();

    document.getElementById('reveal').style.display = 'none';

}
```

This function also hides the original button, leaving only the newly revealed text via setting display to'none'. Now, add this new function back to your HTML in this line

<button id="reveal" onclick="sendTheAJAX()" class="button">Why's that?</button>



Python Introduction:

Python is a general purpose, dynamic, high level and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is *easy to learn* yet powerful and versatile scripting language which makes it attractive for Application Development.

Python's syntax and *dynamic typing* with its interpreted nature, makes it an ideal language for scripting and rapid application development.

Python supports *multiple programming pattern*, including object oriented, imperative and functional or procedural programming styles.

Python is not intended to work on special area such as web programming. That is why it is known as *multipurpose* because it can be used with web, enterprise, 3D CAD etc.

We don't need to use data types to declare variable because it is *dynamically typed* so we can write a=10 to assign an integer value in an integer variable.

Python makes the development and debugging *fast* because there is no compilation step included in python development and edit-test-debug cycle is very fast.

Python Features:

Python provides lots of features that are listed below.

1) Easy to Learn and Use

Python is easy to learn and use. It is developer-friendly and high level programming language.

---

2) Expressive Language

Python language is more expressive means that it is more understandable and readable.

---

3) Interpreted Language

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

---

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

---

5) Free and Open Source

Python language is freely available at offical web address.The source-code is also available. Therefore it is open source.

---

6) Object-Oriented Language

Python supports object oriented language and concepts of classes and objects come into existence.

---

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

---

8) Large Standard Library

Python has a large and broad library and prvides rich set of module and functions for rapid application development.

---

9) GUI Programming Support

Graphical user interfaces can be developed using Python.

---

10) Integrated

It can be easily integrated with languages like C, C++, JAVA etc.

Python Applications:

Python is known for its general purpose nature that makes it applicable in almost each domain of software development. Python as a whole can be used in any sphere of development.

Here, we are specifing applications areas where python can be applied.

1) Web Applications:

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser etc. It also provides Frameworks such as Django, Pyramid, Flask etc to design and delelop web based applications. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

2) Desktop GUI Applications:

Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.

3) Software Development:

Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.

4) Scientific and Numeric:

Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

5) Business Applications:

Python is used to build Bussiness applications like ERP and e-commerce systems. Tryton is a high level application platform.

6) Console Based Application:

We can use Python to develop console based applications. For example: IPython.

7) Audio or Video based Applications:

Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

8) 3D CAD Applications:

To create CAD application Fandango is a real application which provides full features of CAD.

9) Enterprise Applications:

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

10) Applications for Images:

Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

There are several such applications which can be developed using Python

Python Data Types:

Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function which returns the type of the variable passed.

Consider the following example to define the values of different data types and checking its type.

```
A=10

b="Hi Python"

c = 10.5

print(type(a));

print(type(b));

print(type(c));
```

Output:

```
<type 'int'>

<type 'str'>

<type 'float'>
```

Standard data types:

A variable can hold different types of values. For example, a person?s name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

- Numbers
- String
- List
- Tuple
- Dictionary

In this section of the tutorial, we will give a brief introduction of the above data types. We will discuss each one of them in detail later in this tutorial.

Numbers:

Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;

```
1. a = 3 , b = 5  #a and b are number objects
```

Python supports 4 types of numeric data.

1. int (signed integers like 10, 2, 29, etc.)
2. long (long integers used for a higher range of values like 908090800L, -0x1929292L, etc.)
3. float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
4. complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

Python allows us to use a lower-case L to be used with long integers. However, we must always use an upper-case L to avoid confusion.

A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts respectively).

String:

The string can be defined as the sequence of characters represented in the quotation marks. In python, we can use single, double, or triple quotes to define a string.

String handling in python is a straightforward task since there are various inbuilt functions and operators provided.

In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.

The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".

```
1. str1 = 'hello javatpoint' #string str1
2. str2 = ' how are you' #string str2
3. print (str1[0:2]) #printing first two character using slice operator
4. print (str1[4]) #printing 4th character of the string
5. print (str1*2) #printing the string twice
6. print (str1 + str2) #printing the concatenation of str1 and str2
```

Output:

he

hello javatpointhello javatpoint

hello javatpoint how are you

List

Lists are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

Consider the following example.

```
1. l  = [1, "hi", "python", 2]
2. print (l[3:]);
3. print (l[0:2]);
4. print (l);
5. print (l + l);
6. print (l * 3);
```

Output:

```
[2]
[1, 'hi']
[1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
```

Tuple:

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Let's see a simple example of the tuple.

```
1. t  = ("hi", "python", 2)
2. print (t[1:]);
3. print (t[0:1]);
4. print (t);
5. print (t + t);
6. print (t * 3);
7. print (type(t))
8. t[2] = "hi";
```

Output:

```
('python', 2)
('hi',)
('hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2, 'hi', 'python', 2)
<type 'tuple'>
Traceback (most recent call last):
  File "main.py", line 8, in <module>
    t[2] = "hi";
TypeError: 'tuple' object does not support item assignment
```

Dictionary:

Dictionary is an ordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type whereas value is an arbitrary Python object.

The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

Consider the following example.

```
1. d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};
2. print("1st name is "+d[1]);
3. print("2nd name is "+ d[4]);
4. print (d);
5. print (d.keys());
6. print (d.values());
```

Output:

```
1st name is Jimmy
2nd name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
[1, 2, 3, 4]
['Jimmy', 'Alex', 'john', 'mike']
```

Till now, we were taking the input from the console and writing it back to the console to interact with the user.

Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again.

However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling.

In this section of the tutorial, we will learn all about file handling in python including, creating a file, opening a file, closing a file, writing and appending the file, etc.

Opening a file:

Python provides the open() function which accepts two arguments, file name and access mode in which the file is accessed. The function returns a file object which can be used to perform various operations like reading, writing, etc.

The syntax to use the open() function is given below.

file object = open(<file-name>, <access-mode>, <buffering>)

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

SN	Access mode	Description

1	R	It opens the file to read-only. The file pointer exists at the beginning. The file is by default open in this m is passed.
2	Rb	It opens the file to read only in binary format. The file pointer exists at the beginning of the file.
3	r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4	rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file
5	w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exis The file pointer exists at the beginning of the file.
6	wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new with the same name. The file pointer exists at the beginning of the file.
7	w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous f r+ doesn?t overwrite the previously written file. It creates a new file if no file exists. The file pointer exis the file.
8	wb+	It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file
9	a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exis file if no file exists with the same name.
10	ab	It opens the file in the append mode in binary format. The pointer exists at the end of the previously writt file in binary format if no file exists with the same name.
11	a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It cre exists with the same name.
12	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

Let's look at the simple example to open a file named "file.txt" (stored in the same directory) in read mode and printing its content on the console.

Example:

```
1. #opens the file file.txt in read mode
2. fileptr = open("file.txt","r")
3.
4. if fileptr:
5.     print("file is opened successfully")
```

Output:

```
<class '_io.TextIOWrapper'>
file is opened successfully
```

The close() method:

Once all the operations are done on the file, we must close it through our python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.

We can perform any operation on the file externally in the file system is the file is opened in python, hence it is good practice to close the file once all the operations are done.

The syntax to use the close() method is given below.

fileobject.close()

Consider the following example.

Example

```
1. # opens the file file.txt in read mode
2. fileptr = open("file.txt","r")
3.
4. if fileptr:
5.     print("file is opened successfully")
6.
7. #closes the opened file
8. fileptr.close()
```

Reading the file:

To read a file using the python script, the python provides us the read() method. The read() method reads a string from the file. It can read the data in the text as well as binary format.

The syntax of the read() method is given below.

fileobj.read(<count>)

Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.

Consider the following example.

Example:

```
1. #open the file.txt in read mode. causes error if no such file exists.
2. fileptr = open("file.txt","r");
3.
4. #stores all the data of the file into the variable content
5. content = fileptr.read(9);
6.
7. # prints the type of the data stored in the file
8. print(type(content))
9. #prints the content of the file
10. print(content)
      #closes the opened file
11. fileptr.close()
```

Output:

```
<class 'str'>
Hi, I am
```

Read Lines of the file:

Python facilitates us to read the file line by line by using a function readline(). The readline() method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.

Consider the following example which contains a function readline() that reads the first line of our file **"file.txt"** containing three lines.

Example

```
1. #open the file.txt in read mode. causes error if no such file exists.
2. fileptr = open("file.txt","r");
3.
4. #stores all the data of the file into the variable content
5. content = fileptr.readline();
6.
7. # prints the type of the data stored in the file
```

```
8. print(type(content))
9.
10. #prints the content of the file
11. print(content)
12.
13. #closes the opened file
14. fileptr.close()
```

Output:

```
<class 'str'>
Hi, I am the file and being used as
```

Looping through the file:

By looping through the lines of the file, we can read the whole file.

Example

#open the file.txt in read mode. causes an error if no such file exists.

```
fileptr = open("file.txt","r");

#running a for loop
for i in fileptr:
   print(i) # i contains each line of the file
```

Output:

```
Hi, I am the file and being used as
an example to read a
file in python.
```

Writing the file:

To write some text to a file, we need to open the file using the open method with one of the following access modes.

a: It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.

w: It will overwrite the file if any file exists. The file pointer is at the beginning of the file.

Consider the following example.

Example 1

```
#open the file.txt in append mode. Creates a new file if no such file exists.
fileptr = open("file.txt","a");

#appending the content to the file
fileptr.write("Python is the modern day language. It makes things so simple.")


#closing the opened file
fileptr.close();
```

Now, we can see that the content of the file is modified.

File.txt:

```
Hi, I am the file and being used as
an example to read a
file in python.
Python is the modern day language. It makes things so simple.
```

Example 2

```
#open the file.txt in write mode.
fileptr = open("file.txt","w");

#overwriting the content of the file
fileptr.write("Python is the modern day language. It makes things so simple.")


#closing the opened file
fileptr.close();
```

Now, we can check that all the previously written content of the file is overwritten with the new text we have passed.

File.txt:

```
Python is the modern day language. It makes things so simple.
```

Creating a new file:

The new file can be created by using one of the following access modes with the function open(). **x:** it creates a new file with the specified name. It causes an error a file exists with the same name.

a: It creates a new file with the specified name if no such file exists. It appends the content to the file if the file already exists with the specified name.

w: It creates a new file with the specified name if no such file exists. It overwrites the existing file.

Consider the following example.

Example

```
1. #open the file.txt in read mode. causes error if no such file exists.
2. fileptr = open("file2.txt","x");
3.
4. print(fileptr)
5.
6. if fileptr:
7.     print("File created successfully");
```

Output:

```
File created successfully
```

Using with statement with files:

The with statement was introduced in python 2.5. The with statement is useful in the case of manipulating the files. The with statement is used in the scenario where a pair of statements is to be executed with a block of code in between.

The syntax to open a file using with statement is given below.

```
with open(<file name>, <access mode>) as <file-pointer>:

    #statement suite
```

The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the with statement in the case of file s because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file. It doesn't let the file to be corrupted.

Consider the following example.

Example:

```
1. with open("file.txt",'r') as f:
2.     content = f.read();
3.     print(content)
```

Output:

Python is the modern day language. It makes things so simple.

File Pointer positions:

Python provides the tell() method which is used to print the byte number at which the file pointer exists. Consider the following example.

Example

```
1. # open the file file2.txt in read mode
2. fileptr = open("file2.txt","r")
3.
4. #initially the filepointer is at 0
5. print("The filepointer is at byte :",fileptr.tell())
6.
7. #reading the content of the file
8. content = fileptr.read();
9.
10. #after the read operation file pointer modifies. tell() returns the location of the fileptr.
11.
12. print("After reading, the filepointer is at:",fileptr.tell())
```

Output:

```
The filepointer is at byte : 0
After reading, the filepointer is at 26
```

Modifying file pointer position:

In the real world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations.

For this purpose, the python provides us the seek() method which enables us to modify the file pointer position externally.

The syntax to use the seek() method is given below.

<file-ptr>.seek(offset[, **from**)

The seek() method accepts two parameters:

offset: It refers to the new position of the file pointer within the file.

from: It indicates the reference position from where the bytes are to be moved. If it is set to 0, the beginning of the file is used as the reference position. If it is set to 1, the current position of the file pointer is used as the reference position. If it is set to 2, the end of the file pointer is used as the reference position.

Consider the following example.

Example

```
1. # open the file file2.txt in read mode
2. fileptr = open("file2.txt","r")
3.
4. #initially the filepointer is at 0
5. print("The filepointer is at byte :",fileptr.tell())
6.
7. #changing the file pointer location to 10.
8. fileptr.seek(10);
9.
10. #tell() returns the location of the fileptr.
11. print("After reading, the filepointer is at:",fileptr.tell())
```

Output:

```
The filepointer is at byte : 0
After reading, the filepointer is at 10
```

Python os module:

The os module provides us the functions that are involved in file processing operations like renaming, deleting, etc.

Let's look at some of the os module functions.

Renaming the file:

The os module provides us the rename() method which is used to rename the specified file to a new name. The syntax to use the rename() method is given below.

rename(?current-name?, ?new-name?)

Example

```
1. import os;
2.
3. #rename file2.txt to file3.txt
4. os.rename("file2.txt","file3.txt")
```

Removing the file:

The os module provides us the remove() method which is used to remove the specified file. The syntax to use the remove() method is given below.

1. remove(?file-name?)

Example:

```
import os;

#deleting the file named file3.txt
os.remove("file3.txt")
```

Creating the new directory:

The mkdir() method is used to create the directories in the current working directory. The syntax to create the new directory is given below.

mkdir(?directory name?)

Example:

```
1. import os;
2.
3. #creating a new directory with the name new
4. os.mkdir("new")
```

Changing the current working directory:

The chdir() method is used to change the current working directory to a specified directory.

The syntax to use the chdir() method is given below.

1. chdir("new-directory")

Example

```
1. import os;
2.
3. #changing the current working directory to new
4.
5. os.chdir("new")
```

The getcwd() method:

This method returns the current working directory.

The syntax to use the getcwd() method is given below.

1. os.getcwd()

Example

```
1. import os;
2.
3. #printing the current working directory
4. print(os.getcwd())
```

Deleting directory:

The rmdir() method is used to delete the specified directory.

The syntax to use the rmdir() method is given below.

1. os.rmdir(?directory name?)

Example

```
1. import os;
2.
3. #removing the new directory
4. os.rmdir("new")
```

Writing python output to the files:

In python, there are the requirements to write the output of a python script to a file.

The **check_call()** method of module **subprocess** is used to execute a python script and write the output of that script to a file.

The following example contains two python scripts. The script file1.py executes the script file.py and writes its output to the text file **output.txt**

file.py:

```
1. temperatures=[10,-20,-289,100]
2. def c_to_f(c):
3.     if c< -273.15:
4.         return "That temperature doesn't make sense!"
5.     else:
6.         f=c*9/5+32
7.         return f
8. for t in temperatures:
9.     print(c_to_f(t))
```

file.py:

```
1. import subprocess
2.
3. with open("output.txt", "wb") as f:
4.     subprocess.check_call(["python", "file.py"], stdout=f)
```

Output:

```
50
-4
That temperature doesn't make sense!
212
```

The file related methods:

The file object provides the following methods to manipulate the files on various operating systems.

SN	Method	Description
1	file.close()	It closes the opened file. The file once closed, it can?t be read or write any more.
2	File.fush()	It flushes the internal buffer.
3	File.fileno()	It returns the file descriptor used by the underlying implementation to request I/O from the OS.
4	File.isatty()	It returns true if the file is connected to a TTY device, otherwise returns false.
5	File.next()	It returns the next line from the file.
6	File.read([size])	It reads the file for the specified size.
7	File.readline([size])	It reads one line from the file and places the file pointer to the beginning of the new line.

8	File.readlines([sizehint])	It returns a list containing all the lines of the file. It reads the file until the EOF occurs using readline() function.
9	File.seek(offset[,from)	It modifies the position of the file pointer to a specified offset with the specified reference.
10	File.tell()	It returns the current position of the file pointer within the file.
11	File.truncate([size])	It truncates the file to the optional specified size.
12	File.write(str)	It writes the specified string to a file
13	File.writelines(seq)	It writes a sequence of the strings to a file.

Python Classes and Objects:

❮ PreviousNext ❯

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword `class`:

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

Create Object

Now we can use the class named myClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

The __init__() Function:

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Note: The __init__() function is called automatically every time the class is being used to create a new object.

Object Methods:

Objects can also contain methods. Methods in objects are functions that belongs to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Note: The self parameter is a reference to the class instance itself, and is used to access variables that belongs to the class.

The self Parameter:

The `self` parameter is a reference to the class itself, and is used to access variables that belongs to the class.

It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

```
class Person:
  def __init__(mysillyobject, name, age):
```

```
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Modify Object Properties:

You can modify properties on objects like this:

Example

Set the age of p1 to 40:

```
p1.age = 40
```

Delete Object Properties:

You can delete properties on objects by using the `del` keyword:

Example

Delete the age property from the p1 object:

```
del p1.age
```

Delete Objects:

You can delete objects by using the del keyword:

Example

Delete the p1 object:

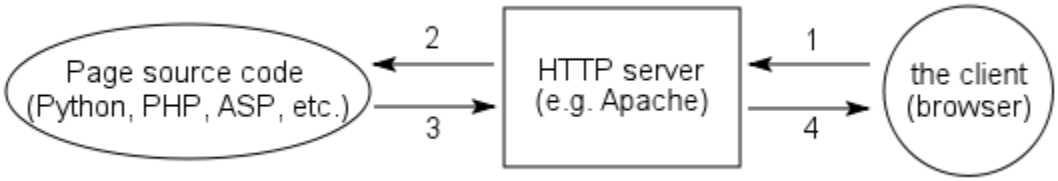
```
del p1
```

DYNAMIC WEB PAGES:

Introduction

The basic idea behind dynamic pages is very simple - instead of preparing all necessary pages as individual files, write a program that will create the pages as they are requested by the user. The program itself can be do whatever you want, the only limitation is that it should produce a HTML page as a result of its execution.

The following schema shows how a dynamic page is processed



1. The client asks the server for a specific page
2. The server finds the file and according to some rules in its configuration determines that it is a dynamic page - that is it should not return the document itself but rather run it to obtain the resulting document.
3. The server executes the source of the page (more on this below) and reads the output.
4. The server sends the output to the client. For the client it is undistinguishable from a static page.

The most interesting part of the whole schema is part number 3 - the execution of the script. In the simplest case of CGI(In computing, **Common Gateway Interface** (**CGI**) offers a standard protocol for web servers to execute programs that execute like console applications (also called command-line interface programs) running on a server that generates web pages dynamically.), there is a program associated with the page that is responsible for generating the resulting document. In more sophisticated cases, an interpreter of the source code is build directly into the web-server (either hard-coded or as a module) and takes care of the execution inside the server.
In case of Python, the most commonly used way to create dynamic pages is to use *mod_python* module for the Apache web-server. This module creates an extra layer between the server and the script which simplifies creation of dynamic pages.

Mod_python offers a few different ways to write server-side scripts, using the so called *handlers*. The most commonly used handler is *publisher* that offers a good mix between flexibility and ease of use.

In scripts interpreted by the *publisher* handler, each function acts as an individual page. This means that if your script is called "test.py", you can access its function "hello" under the URL "test.py/hello". In case you don't supply the name of the function (the URL is just "test.py"), the result of a function called "index" is returned. This way you can create all the pages in one big file, or you can have each page in a separate file with only the "index" function.

The following example demonstrates a very simple dynamic page using the *publisher* handler.

Zdroj: (dpages1-1.py)

```
1  def index():
2    return "<html><body><h1>Hello World</h1></body></html>"
```

dpages1-1.py.html

```
<html><body><h1>Hello World</h1></body></html>
```

Simple example

The following example shows a simple dynamic web-page created using the *publisher* handler of mod_python.

The function *index* is run by default if no relative path is provided after the script name. The function can be declared as having no arguments or it can have a **req** argument that will contain a Request object. We will not use this object in the following examples, but it contains all the information from the request HTTP headers - the URL, form attributes, etc. and might be useful in more complicated scripts.

Zdroj: (dpages2-1.py)

```
1  page_template = """
```

```
<html>
<head>
<title>%s</title>
</head>
<body>
%s
</body>
</html>"""


def index( req):
  title = "Current time"
  import datetime
  body = "Current time is: "+datetime.datetime.now().strftime( "%Y-%m-%d %H:%M:%S")
  return page_template % (title, body)
```

dpages2-1.py.html

```
<html>
</head>
<body>
Current time is: 2008-03-12 10:56:37
</body>
</html>
```

If a relative path is given, it selects a function inside the script.

URL:

dpages2-2.py/test

Zdroj: (dpages2-2.py)

```
page_template = """
<html>
<head>
<title>%s</title>
</head>
```

```
<body>
%s
</body>
</html>"""


def index():
  title = "Current time"
  import datetime
  body = "Current time is: "+datetime.datetime.now().strftime( "%Y-%m-%d %H:%M:%S")
  return page_template % (title, body)

def test():
  return page_template % ("Test", "This is a test page")
```

dpages2-2.html

```
<html>
<head>
<title>Test</title>
</head>
<body>
This is a test page
</body>
</html>
```

Adding *index* as the relative path is the same as not giving it.

URL:

```
dpages2-2.py/index
```

Zdroj: (dpages2-2.py)

```
page_template = """
<html>
```

```
<head>
<title>%s</title>
</head>
<body>
%s
</body>
</html>"""


def index():
  title = "Current time"
  import datetime
  body = "Current time is: "+datetime.datetime.now().strftime( "%Y-%m-%d %H:%M:%S")
  return page_template % (title, body)

def test():
  return page_template % ("Test", "This is a test page")
```

dpages2-2.html

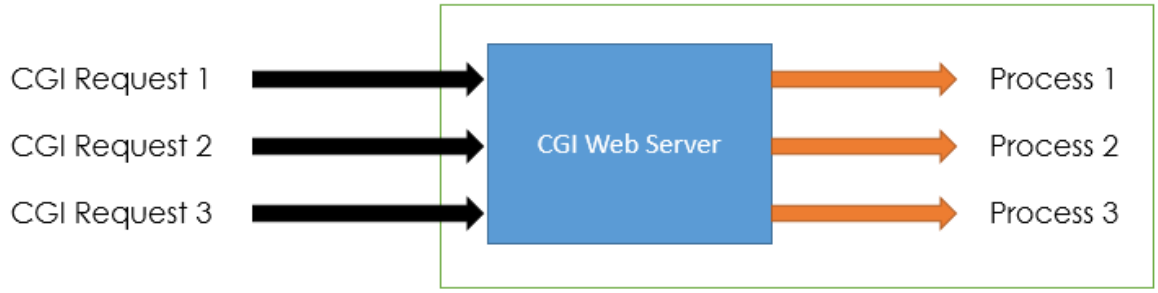
```
<html>
<head>
<title>Current time</title>
</head>
<body>
Current time is: 2008-03-12 10:56:37
</body>
</html>
```

UNIT IV

CGI (Commmon Gateway Interface)

1. CGI (Common Gateway Interface) is used to provide dynamic content to the user.
2. CGI is used to execute a program that resides in the server to process data or access databases to produce the relevant dynamic content.
3. Programs that resides in server can be written in native operating system such as C++.

Diagrammatic Representation :



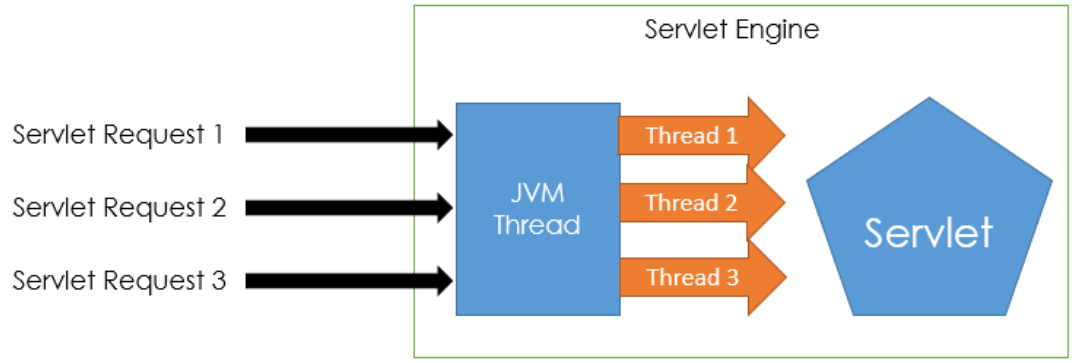
We have listed some problems in CGI technology –

Disadvantages of CGI :

1. For each request CGI Server receives, It creates new Operating System Process.
2. If the number of requests from the client increases then more time it will take to respond to the request.
3. As programs executed by CGI Script are written in the native languages such as C, C++, perl which are platform independent.

Servlet :

CGI programs are used to execute programs written inside the native language. But in Servlet all the programs are compiled into the Java bytecode which is then run in the Java virtual machine.



In Servlet, All the requests coming from the Client are processed with the threads instead of the OS process.

Servlet Vs CGI :

Let's differentiate Servlet and CGI –

Servlet	**CGI (Common Gateway Interface)**
Servlets are portable	CGI is not portable.
In Servlets each request is handled by lightweight Java Thread	IN CGI each request is handled by heavy weight OS process
In Servlets, Data sharing is possible	In CGI, data sharing is not available.
Servlets can link directly to the Web server	CGI cannot directly link to Web server.
Session tracking and caching of previous computations can be performed	Session tracking and caching of previous computations cannot be performed
Automatic parsing and decoding of HTML form data can be performed.	Automatic parsing and decoding of HTML form data cannot be performed.
Servlets can read and Set HTTP Headers	CGI cannot read and Set HTTP Headers
Servlets can handle cookies	CGI cannot handle cookies

Servlet	**CGI (Common Gateway Interface)**
Servlets can track sessions	CGI cannot track sessions
Servlets is inexpensive than CGI	CGI is more expensive than Servlets

Static vs Dynamic website

Static Website	**Dynamic Website**
Prebuilt content is same every time the page is loaded.	Content is generated quickly and changes regularly.
It uses the **HTML** code for developing a website.	It uses the server side languages such as **PHP,SERVLET, JSP, and ASP.NET** etc. for developing a website.
It sends exactly the same response for every request.	It may generate different HTML for each of the request.
The content is only changed when someone publishes and updates the file (sends it to the web server).	The page contains "server-side" code which allows the server to generate the unique content when the page is loaded.
Flexibility is the main advantage of static website.	Content Management System (CMS) is the main advantage of dynamic website.

Java servlet:

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

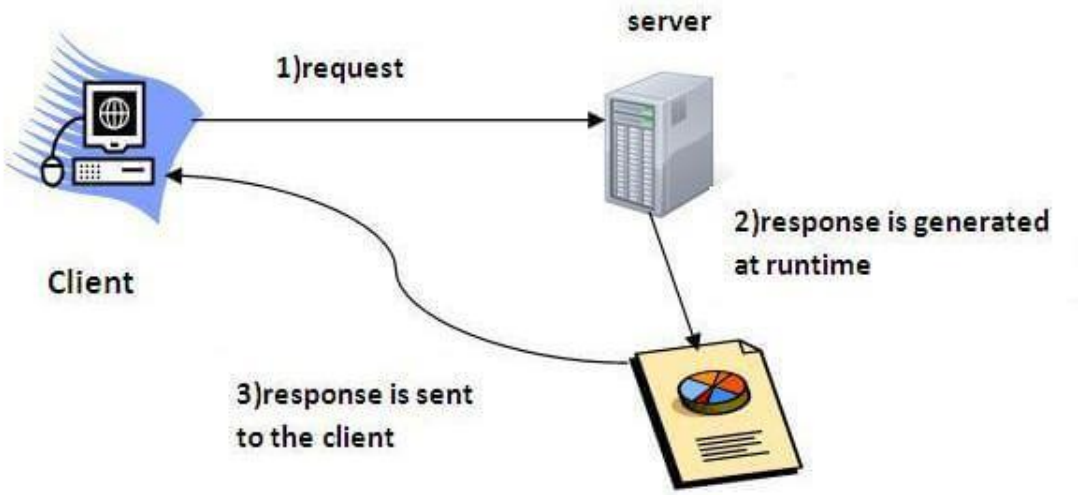
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

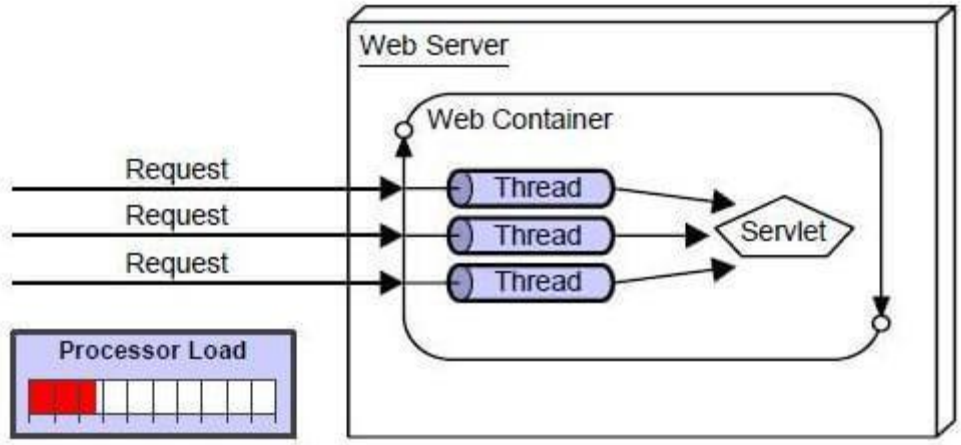
What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.



Benifits of Servlet



There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.

3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

Life cycle of java servlet:

Life Cycle of a Servlet

The entire life cycle of a Servlet is managed by the Servlet container which uses the **javax.servlet.Servlet** interface to understand the Servlet object and manage it. So, before creating a Servlet object let's first understand the life cycle of the Servlet object which is actually understanding that how the Servlet container manages the Servlet object.

Stages of the Servlet Life Cycle: The Servlet life cycle mainly goes through four stages,

Loading a Servlet.

Initializing the Servlet.

Request handling.

Destroying the Servlet.

Let's look at each of these stages in details:

Loading a Servlet: The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages :

-Initializing the context, on configuring the Servlet with a zero or positive integer value.

-If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

Loading : Loads the Servlet class.

Instantiation : Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.

Initializing a Servlet: After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts ServletConfig object reference as parameter.

The Servlet container invokes the **Servlet.init(ServletConfig)** method only once, immediately after the **Servlet.init(ServletConfig)** object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.

Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the **ServletException** or **UnavailableException.**

Handling request: After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request :

It creates the **ServletRequest** and **ServletResponse** objects. In this case, if this is a HTTP request then the Web container creates **HttpServletRequest** and **HttpServletResponse** objects which are subtypes of the **ServletRequest** and **ServletResponse** objects respectively.

After creating the request and response objects it invoke the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

The **service()** method while processing the request may throw the **ServletException** or **UnavailableException** or **IOException.**

Destroying a Servlet: When a Servlet container decides to destroy the Servlet, it performs the following operations,

It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.

After currently running threads have completed their jobs, the Servlet container calls the **destroy()** method on the Servlet instance.

After the **destroy()** method is executed, the Servlet conatiner releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

Working with cookies:

Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name a value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.

public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provide other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is use return all the cookies from the browser.

How to create Cookie?

Let's see the simple code to create cookie.

```
1. Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
2. response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
1. Cookie ck=new Cookie("user","");//deleting value of cookie
2. ck.setMaxAge(0);//changing the maximum age to 0 seconds
3. response.addCookie(ck);//adding cookie in the response
```

How to get Cookies?

Let's see the simple code to get all the cookies.

```
1. Cookie ck[]=request.getCookies();
2. for(int i=0;i<ck.length;i++){
3.  out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
4. }
```

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.

index.html

```
1. <form action="servlet1" method="post">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.  public class FirstServlet extends HttpServlet {
5.  public void doPost(HttpServletRequest request, HttpServletResponse response){
6.    try{
7.    response.setContentType("text/html");
8.    PrintWriter out = response.getWriter();
9.    String n=request.getParameter("userName");
10.   out.print("Welcome "+n);
11.   Cookie ck=new Cookie("uname",n);//creating cookie object
12.   response.addCookie(ck);//adding cookie in the response
13.   //creating submit button
```

```
14.    out.print("<form action='servlet2'>");
15.    out.print("<input type='submit' value='go'>");
16.    out.print("</form>");
17.    out.close();
18.      }catch(Exception e){System.out.println(e);}
19.  }  }
```

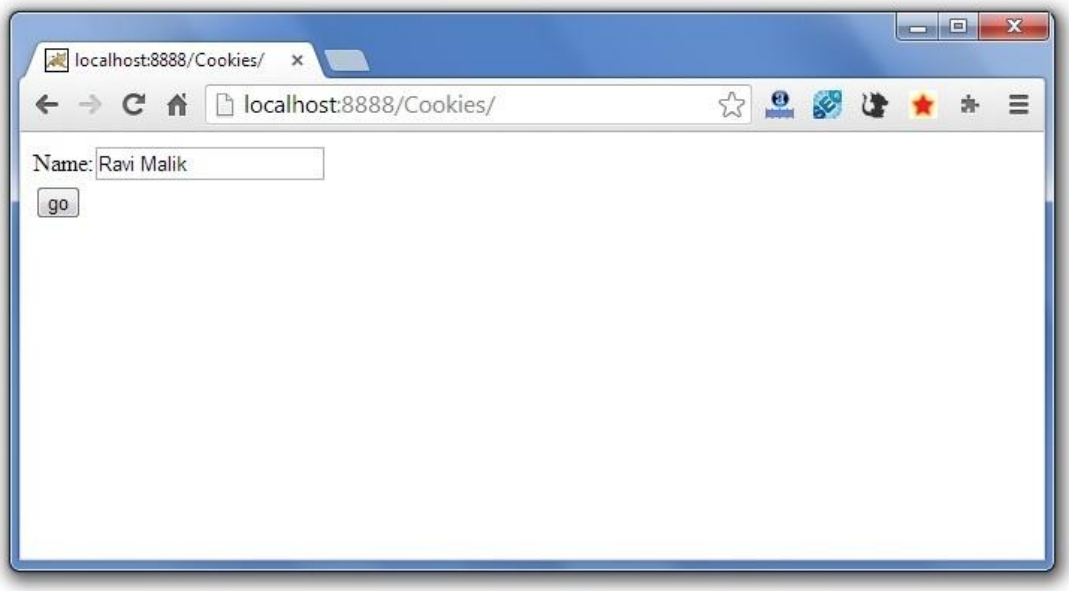
SecondServlet.java

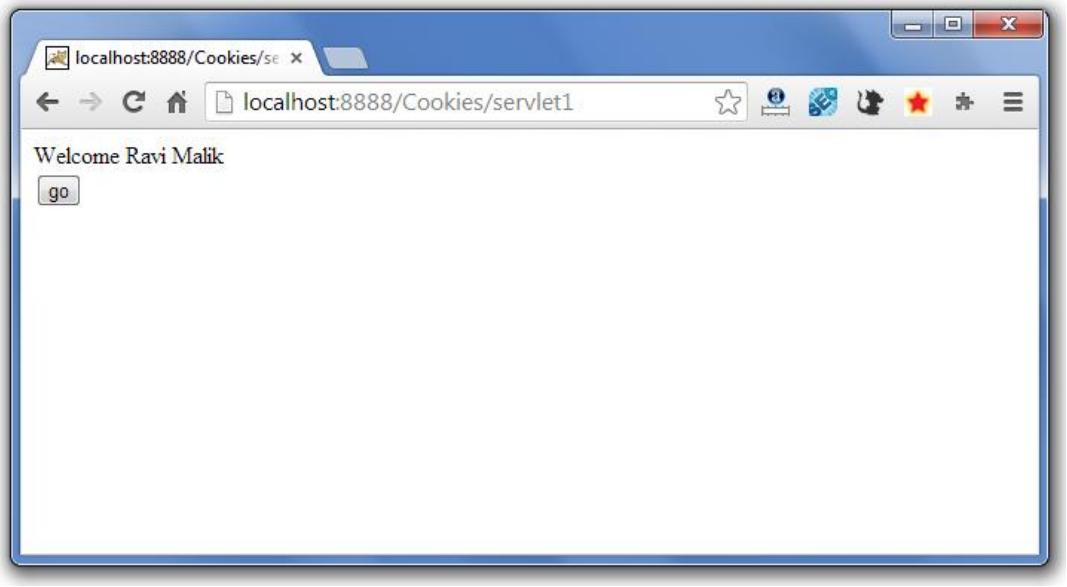
```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.    public class SecondServlet extends HttpServlet {
5.    public void doPost(HttpServletRequest request, HttpServletResponse response){
6.      try{
7.      response.setContentType("text/html");
8.      PrintWriter out = response.getWriter();
9.      Cookie ck[]=request.getCookies();
10.     out.print("Hello "+ck[0].getValue());
11.   out.close();
12.       }catch(Exception e){System.out.println(e);}
13.    }
14.   }
```

web.xml

```
1.  <web-app>
2.    <servlet>
3.  <servlet-name>s1</servlet-name>
4.  <servlet-class>FirstServlet</servlet-class>
5.  </servlet>
6.    <servlet-mapping>
7.  <servlet-name>s1</servlet-name>
8.  <url-pattern>/servlet1</url-pattern>
9.  </servlet-mapping>
10.   <servlet>
11. <servlet-name>s2</servlet-name>
12. <servlet-class>SecondServlet</servlet-class>
13. </servlet>
14.   <servlet-mapping>
```

15. <servlet-name>s2</servlet-name>
16. <url-pattern>/servlet2</url-pattern>
17. </servlet-mapping>
18. </web-app>





Session Tracking in Servlets

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:

Why use Session Tracking?

To recognize the user It is used to recognize the particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

2) Hidden Form Field

In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

```
1. <input type="hidden" name="uname" value="Vimal Jaiswal">
```

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

Real application of hidden form field

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

Example of using Hidden Form Field

In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.

index.html

```
1. <form action="servlet1">
```

```
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.   public class FirstServlet extends HttpServlet {
5.  public void doGet(HttpServletRequest request, HttpServletResponse response){
6.          try{
7.           response.setContentType("text/html");
8.          PrintWriter out = response.getWriter();
9.          String n=request.getParameter("userName");
10.         out.print("Welcome "+n);
11.         //creating form that have invisible textfield
12.         out.print("<form action='servlet2'>");
13.         out.print("<input type='hidden' name='uname' value='"+n+"'>");
14.         out.print("<input type='submit' value='go'>");
15.         out.print("</form>");
16.         out.close();
17.              }catch(Exception e){System.out.println(e);}
18.     }
19.  }
```

SecondServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.  public class SecondServlet extends HttpServlet {
5.  public void doGet(HttpServletRequest request, HttpServletResponse response)
6.          try{
7.          response.setContentType("text/html");
8.          PrintWriter out = response.getWriter();
9.          //Getting the value from the hidden field
10.         String n=request.getParameter("uname");
11.         out.print("Hello "+n);
12.         out.close();
13.              }catch(Exception e){System.out.println(e);}
```

```
14.     }
15. }
```

web.xml

```
1.  <web-app>
2.   <servlet>
3.  <servlet-name>s1</servlet-name>
4.  <servlet-class>FirstServlet</servlet-class>
5.  </servlet>
6.   <servlet-mapping>
7.  <servlet-name>s1</servlet-name>
8.  <url-pattern>/servlet1</url-pattern>
9.  </servlet-mapping>
10.  <servlet>
11. <servlet-name>s2</servlet-name>
12. <servlet-class>SecondServlet</servlet-class>
13. </servlet>
14.  <servlet-mapping>
15. <servlet-name>s2</servlet-name>
16. <url-pattern>/servlet2</url-pattern>
17. </servlet-mapping>
18.   </web-app>
```

3)URL Rewriting

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value

pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.

Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

Example of using URL Rewriting

In this example, we are maintaning the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String n=request.getParameter("userName");
      out.print("Welcome "+n);
```

```
11.       //appending the username in the query string
12.        out.print("<a href='servlet2?uname="+n+"'>visit</a>");
13.         out.close();
14.             }catch(Exception e){System.out.println(e);}
15.    }  }
```

SecondServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.    public class SecondServlet extends HttpServlet {
5.   public void doGet(HttpServletRequest request, HttpServletResponse response)
6.          try{
7.          response.setContentType("text/html");
8.          PrintWriter out = response.getWriter();
9.          //getting value from the query string
10.         String n=request.getParameter("uname");
11.         out.print("Hello "+n);
12.          out.close();
13.               }catch(Exception e){System.out.println(e);}
14.    }  }
```

web.xml

```
1.  <web-app>
2.    <servlet>
3.  <servlet-name>s1</servlet-name>
4.  <servlet-class>FirstServlet</servlet-class>
5.  </servlet>
6.    <servlet-mapping>
7.  <servlet-name>s1</servlet-name>
8.  <url-pattern>/servlet1</url-pattern>
9.  </servlet-mapping>
10.   <servlet>
11. <servlet-name>s2</servlet-name>
12. <servlet-class>SecondServlet</servlet-class>
13. </servlet>
14.  <servlet-mapping>
15. <servlet-name>s2</servlet-name>
16. <url-pattern>/servlet2</url-pattern>
17. </servlet-mapping>
```

18. </web-app>

4) HttpSession interface

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

index.html

```
1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.    public class FirstServlet extends HttpServlet {
5.   public void doGet(HttpServletRequest request, HttpServletResponse response){
6.         try{
7.          response.setContentType("text/html");
8.         PrintWriter out = response.getWriter();
9.          String n=request.getParameter("userName");
10.        out.print("Welcome "+n);
11.         HttpSession session=request.getSession();
12.        session.setAttribute("uname",n);
13.       out.print("<a href='servlet2'>visit</a>");
14.             out.close();
15.           }catch(Exception e){System.out.println(e);}
16.    }  }
```

SecondServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.   public class SecondServlet extends HttpServlet {
```

```
5.  public void doGet(HttpServletRequest request, HttpServletResponse response)
6.        try{
7.          response.setContentType("text/html");
8.        PrintWriter out = response.getWriter();
9.          HttpSession session=request.getSession(false);
10.       String n=(String)session.getAttribute("uname");
11.       out.print("Hello "+n);
12.    out.close();
13.            }catch(Exception e){System.out.println(e);}
14.   }  }
```

web.xml

```
1.  <web-app>
2.   <servlet>
3.  <servlet-name>s1</servlet-name>
4.  <servlet-class>FirstServlet</servlet-class>
5.  </servlet>
6.   <servlet-mapping>
7.  <servlet-name>s1</servlet-name>
8.  <url-pattern>/servlet1</url-pattern>
9.  </servlet-mapping>
10.  <servlet>
11. <servlet-name>s2</servlet-name>
12. <servlet-class>SecondServlet</servlet-class>
13. </servlet>
14.  <servlet-mapping>
15. <servlet-name>s2</servlet-name>
16. <url-pattern>/servlet2</url-pattern>
17. </servlet-mapping>
18.  </web-app>
```

Deployment Descriptor:

In a java web application a file named web.xml is known as deployment descriptor. It is a xml file and is the root element for it. When a request comes web server uses web.xml file to map the URL of the request to the specific code that handle the request.

Sample code of web.xml file:

```
<web-app>

 <servlet>
  <servlet-name>servletName</servlet-name>
  <servlet-class>servletClass</servlet-class>
 </servlet>

 <servlet-mapping>
  <servlet-name>servletName</servlet-name>
  <url-pattern>*.*</url-pattern>
 </servlet-mapping>

</web-app>
```

How web.xml works:

When a request comes it is matched with url pattern in servlet mapping attribute. In the above example all urls mapped with the servlet. You can specify a url pattern according to your need. When url matched with url pattern web server try to find the servlet name in servlet attributes same as in servlet mapping attribute. When match found control is goes to the associated servlet class.

Servlet "Hello World" example by extending HttpServlet class.

HelloWorld.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * This servlet program is used to print "Hello World"
 * on client browser using HttpServlet class.
 * @author java tutorial point
 */
public class HelloWorld extends HttpServlet {
    private static final long serialVersionUID = 1L;

    //no-argument constructor.
    public HelloWorld() {

    }

    protected void doGet(HttpServletRequest request,
HttpServletResponse
            response) throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
```

```
        out.println("
```

Hello World using HttpServlet class.

```
");
        out.close();
    }
}
```

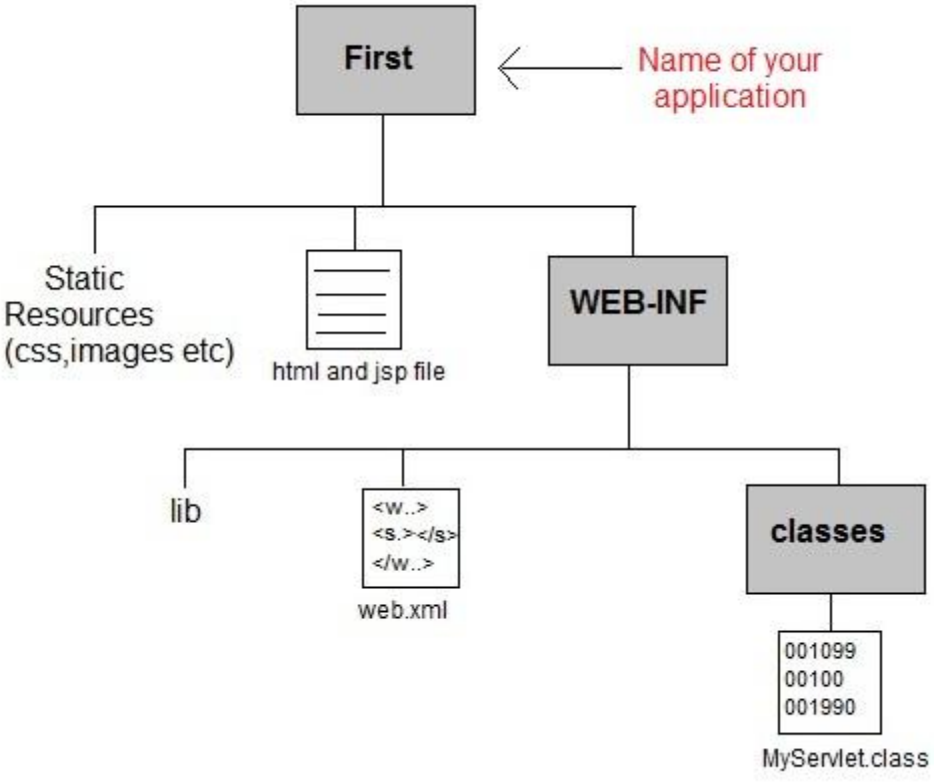
web.xml

```
xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

 <servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>
    com.javawithease.business.HelloWorld
  </servlet-class>
 </servlet>

 <servlet-mapping>
  <servlet-name>HelloWorld</servlet-name>
  <url-pattern>/HelloWorld</url-pattern>
 </servlet-mapping>

</web-app>
```

Output:

Creating the Directory Structure

Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.



Create the above directory structure inside **Apache-Tomcat\webapps** directory. All HTML, static files(images, css etc) are kept directly under **Web application** folder. While all the Servlet classes are kept inside `classes` folder.

The `web.xml` (deployement descriptor) file is kept under `WEB-INF` folder.

servlet API

http://ecomputernotes.com/servlet/intro/servlet-api

SERVLETS - LIFE CYCLE

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init** method.
- The servlet calls **service** method to process a client's request.
- The servlet is terminated by calling the **destroy** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

The init Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this –

```
public void init() throws ServletException {
   // Initialization code...
}
```

The service Method

The service method is the main method to perform the actual task. The servlet container $i.e. webserver$ calls the service method to handle requests coming from the client$browsers$ and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service method checks the HTTP request type $GET, POST, PUT, DELETE, etc.$ and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method –

```
public void service(ServletRequest request, ServletResponse response)
   throws ServletException, IOException {
}
```

The service method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service method but you override either doGet or doPost depending on what type of request you receive from the client.

The doGet and doPost are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   // Servlet code
}
```

The doPost Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   // Servlet code
}
```

The destroy Method

The destroy method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

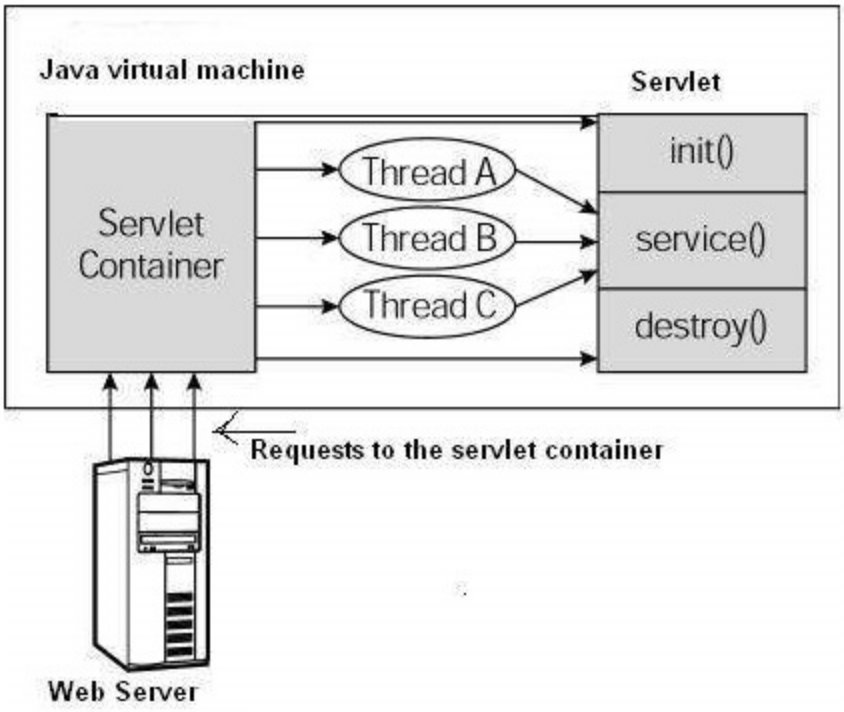
After the destroy method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this –

```
public void destroy() {
   // Finalization code...
}
```

Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service method of a single instance of the servlet.


Java virtual machine
Servlet
Servlet Container
Thread A
Thread B
Thread C
init()
service()
destroy()
Requests to the servlet container
Web Server

SERVLETS - FORM DATA

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** *questionmark* symbol as follows –

```
http://www.test.com/hello?key1 = value1&key2 = value2
```

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.

This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet** method.

POST Method

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? *questionmark* in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost** method.

Reading Form Data using Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation –

- **getParameter** – You call request.getParameter method to get the value of a form parameter.
- **getParameterValues** – Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames** – Call this method if you want a complete list of all parameters in the current request.

GET Method Example using URL

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name = ZARA&last_name = ALI

Given below is the **HelloForm.java** servlet program to handle input given by web browser. We are going to use **getParameter** method which makes it very easy to access passed information –

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>First Name</b>: "
                  + request.getParameter("first_name") + "\n" +
                  "  <li><b>Last Name</b>: "
                  + request.getParameter("last_name") + "\n" +
               "</ul>\n" +
            "</body>" +
         "</html>"
      );
   }
}
```

Assuming your environment is set up properly, compile HelloForm.java as follows –

```
$ javac HelloForm.java
```

If everything goes fine, above compilation would produce **HelloForm.class** file. Next you would have to copy this class file in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
   <servlet-name>HelloForm</servlet-name>
   <servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloForm</servlet-name>
   <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

Now type *http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI* in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result –

USING GET METHOD TO READ FORM DATA

- **First Name:** ZARA
- **Last Name:** ALI

GET Method Example Using Form

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

```
<html>
   <body>
      <form action = "HelloForm" method = "GET">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

Keep this HTML in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access *http://localhost:8080/Hello.htm*, here is the actual output of the above form.

First Name: Last Name:

Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

POST Method Example Using Form

Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is **HelloForm.java** servlet program to handle input given by web browser using GET or POST methods.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
```

```
public class HelloForm extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>First Name</b>: "
                  + request.getParameter("first_name") + "\n" +
                  "  <li><b>Last Name</b>: "
                  + request.getParameter("last_name") + "\n" +
               "</ul>\n" +
            "</body>"
         "</html>"
      );
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now compile and deploy the above Servlet and test it using Hello.htm with the POST method as follows –

```
<html>
   <body>
      <form action = "HelloForm" method = "POST">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

Here is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.

First Name: Last Name:

Based on the input provided, it would generate similar result as mentioned in the above examples.

Passing Checkbox Data to Servlet Program

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code, CheckBox.htm, for a form with two checkboxes

```
<html>
   <body>
      <form action = "CheckBox" method = "POST" target = "_blank">
         <input type = "checkbox" name = "maths" checked = "checked" /> Maths
         <input type = "checkbox" name = "physics"  /> Physics
         <input type = "checkbox" name = "chemistry" checked = "checked" />
                                        Chemistry
         <input type = "submit" value = "Select Subject" />
      </form>
   </body>
</html>
```

The result of this code is the following form

Maths Physics Chemistry

Given below is the CheckBox.java servlet program to handle input given by web browser for checkbox button.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class CheckBox extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Reading Checkbox Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>Maths Flag : </b>: "
                  + request.getParameter("maths") + "\n" +
                  "  <li><b>Physics Flag: </b>: "
```

```
                + request.getParameter("physics") + "\n" +
                "  <li><b>Chemistry Flag: </b>: "
                + request.getParameter("chemistry") + "\n" +
             "</ul>\n" +
          "</body>"
       "</html>"
    );
  }

  // Method to handle POST method request.
  public void doPost(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

     doGet(request, response);
  }
}
```

For the above example, it would display following result –

READING CHECKBOX DATA

- **Maths Flag : :** on
- **Physics Flag:** : null
- **Chemistry Flag:** : on

Reading All Form Parameters

Following is the generic example which uses **getParameterNames** method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order

Once we have an Enumeration, we can loop down the Enumeration in standard way by, using *hasMoreElements* method to determine when to stop and using *nextElement* method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class ReadParams extends HttpServlet {

   // Method to handle GET method request.
```

```
   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Reading All Form Parameters";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n" +
         "<table width = \"100%\" border = \"1\" align = \"center\">\n" +
         "<tr bgcolor = \"#949494\">\n" +
            "<th>Param Name</th>"
            "<th>Param Value(s)</th>\n"+
         "</tr>\n"
      );

      Enumeration paramNames = request.getParameterNames();

      while(paramNames.hasMoreElements()) {
         String paramName = (String)paramNames.nextElement();
         out.print("<tr><td>" + paramName + "</td>\n<td>");
         String[] paramValues = request.getParameterValues(paramName);

         // Read single valued data
         if (paramValues.length == 1) {
            String paramValue = paramValues[0];
            if (paramValue.length() == 0)
               out.println("<i>No Value</i>");
               else
               out.println(paramValue);
         } else {
            // Read multiple valued data
            out.println("<ul>");

            for(int i = 0; i < paramValues.length; i++) {
               out.println("<li>" + paramValues[i]);
            }
            out.println("</ul>");
         }
      }
      out.println("</tr>\n</table>\n</body></html>");
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now, try the above servlet with the following form –

```
<html>
   <body>
      <form action = "ReadParams" method = "POST" target = "_blank">
         <input type = "checkbox" name = "maths" checked = "checked" /> Maths
         <input type = "checkbox" name = "physics"  /> Physics
         <input type = "checkbox" name = "chemistry" checked = "checked" /> Chem
         <input type = "submit" value = "Select Subject" />
      </form>
   </body>
</html>
```

Now calling servlet using the above form would generate the following result –

READING ALL FORM PARAMETERS

Param Name	Param Value(s)
maths	on
chemistry	on

You can try the above servlet to read any other form's data having other objects like text box, radio button or drop down box etc.

HTTP - REQUESTS

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

- A Request-line
- Zero or more header (General|Request|Entity) fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
- Optionally a message-body

The following sections explain each of the entities used in an HTTP request message.

Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by space SP characters.

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

Let's discuss each of the parts mentioned in the Request-Line.

Request Method

The request **method** indicates the method to be performed on the resource identified by the given **Request-URI**. The method is case-sensitive and should always be mentioned in uppercase. The following table lists all the supported methods in HTTP/1.1.

S.N.	Method and Description
1	**GET** The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** Same as GET, but it transfers the status line and the header section only.
3	**POST** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4	**PUT**

Replaces all the current representations of the target resource with the uploaded content.

5 **DELETE**

Removes all the current representations of the target resource given by URI.

6 **CONNECT**

Establishes a tunnel to the server identified by a given URI.

7 **OPTIONS**

Describe the communication options for the target resource.

8 **TRACE**

Performs a message loop back test along with the path to the target resource.

Request-URI

The Request-URI is a Uniform Resource Identifier and identifies the resource upon which to apply the request. Following are the most commonly used forms to specify an URI:

```
Request-URI = "*" | absoluteURI | abs_path | authority
```

S.N.	Method and Description
1	The asterisk * is used when an HTTP request does not apply to a particular resource, but to the server itself, and is only allowed when the method used does not necessarily apply to a resource. For example: **OPTIONS * HTTP/1.1**
2	The **absoluteURI** is used when an HTTP request is being made to a proxy. The proxy is requested to forward the request or service from a valid cache, and return the response. For example: **GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1**
3	The most common form of Request-URI is that used to identify a resource on an origin server or gateway. For example, a client wishing to retrieve a resource directly from the origin server would create a TCP connection to port 80 of the host "www.w3.org" and send the following lines: **GET /pub/WWW/TheProject.html HTTP/1.1** **Host: www.w3.org** Note that the absolute path cannot be empty; if none is present in the original URI, it MUST be given as "/" *theserverroot*.

Request Header Fields

We will study General-header and Entity-header in a separate chapter when we will learn HTTP

header fields. For now, let's check what Request header fields are.

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers.Here is a list of some important Request-header fields that can be used based on the requirement:

- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Expect
- From
- Host
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- Proxy-Authorization
- Range
- Referer
- TE
- User-Agent

You can introduce your custom fields in case you are going to write your own custom Client and Web Server.

Examples of Request Message

Now let's put it all together to form an HTTP request to fetch **hello.htm** page from the web server running on tutorialspoint.com

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Here we are not sending any request data to the server because we are fetching a plain HTML page from the server. Connection is a general-header, and the rest of the headers are request headers. The following example shows how to send form data to the server using request message body:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
```

```
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

Here the given URL */cgi-bin/process.cgi* will be used to process the passed data and accordingly, a response will be returned. Here **content-type** tells the server that the passed data is a simple web form data and **length** will be the actual length of the data put in the message body. The following example shows how you can pass plain XML to your web server:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

HTTP - RESPONSES

After receiving and interpreting a request message, a server responds with an HTTP response message:

- A Status-line
- Zero or more header (General|Response|Entity) fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
- Optionally a message-body

The following sections explain each of the entities used in an HTTP response message.

Message Status-Line

A Status-Line consists of the protocol version followed by a numeric status code and its associated textual phrase. The elements are separated by space SP characters.

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

HTTP Version

A server supporting HTTP version 1.1 will return the following version information:

```
HTTP-Version = HTTP/1.1
```

Status Code

The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

S.N.	Code and Description
1	**1xx: Informational** It means the request was received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error**

It means the request contains incorrect syntax or cannot be fulfilled.

5 **5xx: Server Error**

It means the server failed to fulfill an apparently valid request.

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all registered status codes. A list of all the status codes has been given in a separate chapter for your reference.

Response Header Fields

We will study General-header and Entity-header in a separate chapter when we will learn HTTP header fields. For now, let's check what Response header fields are.

The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status- Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

- Accept-Ranges
- Age
- ETag
- Location
- Proxy-Authenticate
- Retry-After
- Server
- Vary
- WWW-Authenticate

You can introduce your custom fields in case you are going to write your own custom Web Client and Server.

Examples of Response Message

Now let's put it all together to form an HTTP response for a request to fetch the **hello.htm** page from the web server running on tutorialspoint.com

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

The following example shows an HTTP response message displaying error condition when the web server could not find the requested page:

```
HTTP/1.1 404 Not Found
```

```
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Connection: Closed
Content-Type: text/html; charset=iso-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>404 Not Found</title>
</head>
<body>
   <h1>Not Found</h1>
   <p>The requested URL /t.html was not found on this server.</p>
</body>
</html>
```

Following is an example of HTTP response message showing error condition when the web server encountered a wrong HTTP version in the given HTTP request:

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Content-Type: text/html; charset=iso-8859-1
Connection: Closed
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>400 Bad Request</title>
</head>
<body>
   <h1>Bad Request</h1>
   <p>Your browser sent a request that this server could not understand.</p>
   <p>The request line contained invalid characters following the protocol string.</p>
</body>
</html>
```

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

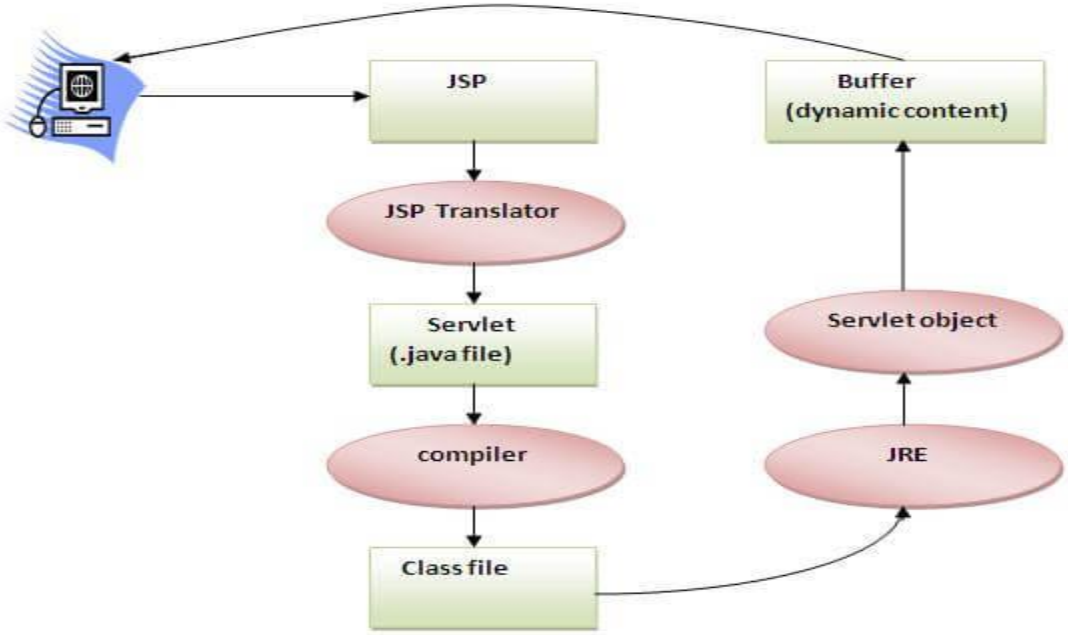
4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy

Creating a simple JSP Page

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

```
1. <html>
2. <body>
3. <% out.print(2*5); %>
4. </body>
5. </html>
```

It will print **10** on the browser.

How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

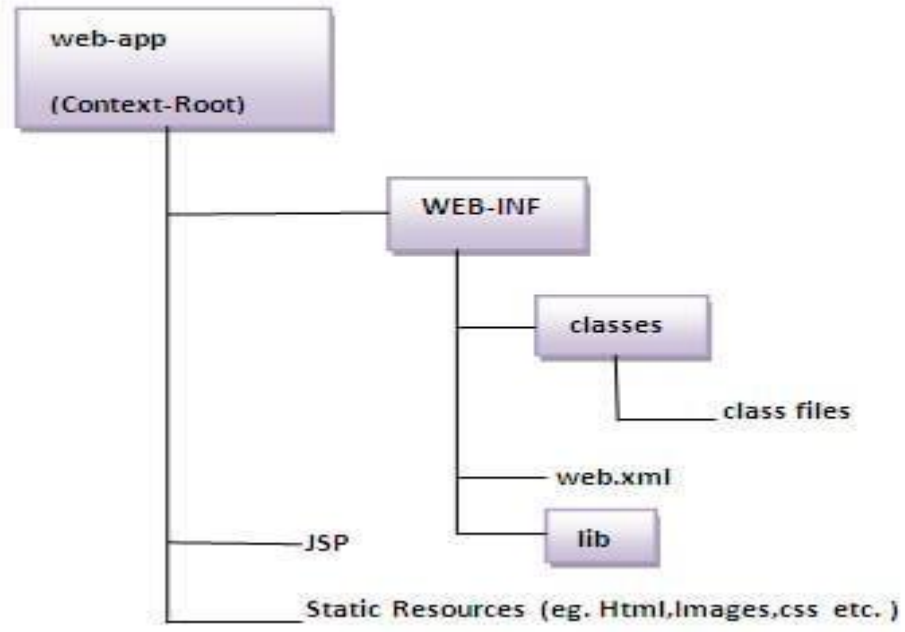
- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

Do I need to follow the directory structure to run a simple JSP?

No, there is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

The Directory structure of JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



The JSP API

The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

javax.servlet.jsp package

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

1. JspPage
2. HttpJspPage

The classes are as follows:

- JspWriter
- PageContext
- JspFactory
- JspEngineInfo
- JspException
- JspError

The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.

Methods of JspPage interface

1. **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

The HttpJspPage interface

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

Method of HttpJspPage interface:

1. **public void _jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.

Introduction:

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages:

1. Extension to Servlet
2. Easy to maintain
3. Less code than Servlet
4. Less Development: No need to recompile and redeploy

JSP directives

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

- page directive
- include directive
- taglib directive

Syntax of JSP Directive

1. <%@ directive attribute="value" %>

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

1. <%@ page attribute="value" %>

Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

1)import

The import attribute is used to import class,interface or all the members of a package.It is similar to imp
class or interface.

Example of import attribute

```
1. <html>
2. <body>
3.
4. <%@ page import="java.util.Date" %>
5. Today is: <%= new Date() %>
6.
7. </body>
8. </html>
```

2)contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

Example of contentType attribute

```
1. <html>
2. <body>
3.
4. <%@ page contentType=application/msword %>
5. Today is: <%= new java.util.Date() %>
6.
7. </body>
8. </html>
```

3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

4)info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

Example of info attribute

```
1. <html>
2. <body>
3.
4. <%@ page info="composed by Sonoo Jaiswal" %>
5. Today is: <%= new java.util.Date() %>
6.
7. </body>
8. </html>
```

The web container will create a method getServletInfo() in the resulting servlet.For example:

```
1. public String getServletInfo() {
2.   return "composed by Sonoo Jaiswal";
3. }
```

---

5)buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

Example of buffer attribute

```
1. <html>
2. <body>
3.
4. <%@ page buffer="16kb" %>
5. Today is: <%= new java.util.Date() %>
6.
7. </body>
8. </html>
```

---

6)language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

---

7)isELIgnored

We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is f

Language is enabled by default. We see Expression Language later.

```
1. <%@ page isELIgnored="true" %>//Now EL will be ignored
```

---

8)isThreadSafe

Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use is of page directive.The value of isThreadSafe value is true.If you make it false, the web container will seri requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to value of isThreadSafe attribute like:

```
<%@ page isThreadSafe="false" %>
```

The web container in such a case, will generate the servlet as:

```
1. public class SimplePage_jsp extends HttpJspBase
2.   implements SingleThreadModel{
3. .......
4. }
```

---

9)errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example of errorPage attribute

```
1. //index.jsp
2. <html>
3. <body>
4.
5. <%@ page errorPage="myerrorpage.jsp" %>
6.
7.  <%= 100/0 %>
8.
9. </body>
10. </html>
```

---

10)isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

```
1. //myerrorpage.jsp
2. <html>
3. <body>
4.
5. <%@ page isErrorPage="true" %>
6.
7.  Sorry an exception occured!<br/>
8. The exception is: <%= exception %>
9.
10. </body>
11. </html>
```

TAGS in JSP:

Declaration tag:
Whenever we use any variables as a part of JSP we have to use those variables in the form of declaration tag i.e., declaration tag is used for declaring the variables in JSP page.

Syntax:

<%! Variable declaration or method definition %>

Example-1:

<%! int a = 10, b = 30, c;%>
<%! int a = 10, b = 30, c;%>

Expression tag:
Expression tags are used for writing the java valid expressions as a part of JSP page.

Syntax:

<%= java valid expression %>

Example-1:

<%! int a = 10, b = 20 %>
<%= a + b%>

Scriplet tag:
Scriplets are basically used to write a pure java code. Whatever the java code we write as a part of scriplet, that code will be available as a part of service () method of servlet.

Syntax:

<% pure java code%>

Implicit Objects in JSP:

These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.

Following list is the nine Implicit Objects that JSP supports −

1. request:

This is the HttpServletRequest object associated with the request.

2. response:

This is the HttpServletResponse object associated with the response to the client.

3. out:

This is the PrintWriter object used to send output to the client.

4. session:

This is the HttpSession object associated with the request.

5. application:

This is the ServletContext object associated with the application context.

6. config:

This is the ServletConfig object associated with the page.

7. pageContext:

This encapsulates use of server-specific features like higher performance JspWriters.

8. page:

This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.

9. exception:

The Exception object allows the exception data to be accessed by designated JSP.

JSP Methods:

JSP Declaration represents a global area for the whole JSP file where programmer can declare variables and methods that can be used throughout JSP code. That is, global variables and methods are declared in Declaration tag.

Example:

```
<HTML>
  <HEAD>
    <TITLE>Creating a Method in jsp</TITLE>
  </HEAD>
<BODY>
    <H1>Creating a Method in jsp</H1>
    <%!
    Int Double(int number)
    {
```

```
    return 2*number;
  }
  %>

  <%
  out.println("The Double of 3 is = " + Double(3));
  %>
 </BODY>
</HTML>
```

Control-Flow Statements:

You can use all the APIs and building blocks of Java in your JSP programming including decision-making statements, loops, etc.

```
<%! int day = 3; %>
<html>
  <head><title>IF...ELSE Example</title></head>

  <body>
    <% if (day == 1 || day == 7) { %>
      <p> Today is weekend</p>
    <% } else { %>
      <p> Today is not weekend</p>
    <% } %>
  </body>
</html>
```

Loop Statements:

You can also use three basic types of looping blocks in Java: for, while, and do…while blocks in your JSP programming.

```
<%! int fontSize; %>
<html>
  <head><title>FOR LOOP Example</title></head>

  <body>
    <%for ( fontSize = 1; fontSize<= 3; fontSize++){ %>
```

```
        <font color = "green" size = "<%= fontSize %>">
          JSP Tutorial
      </font><br />
      <%}%>
   </body>
</html>
```

JSP request implicit object:

The JSP request is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

Example of JSP request implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

JSP Session

In this chapter, we will discuss session tracking in JSP. HTTP is a "stateless" protocol which means each time a client retrieves a Webpage, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Maintaining Session Between Web Client And Server
Let us now discuss a few options to maintain the session between the Web Client and the Web Server −

Cookies
A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.

Hidden Form Fields
A web server can send a hidden HTML form field along with a unique session ID as follows −

<input type = "hidden" name = "sessionid" value = "12345">
This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or the POST data. Each time the web browser sends the request back, the session_id value can be used to keep the track of different web browsers.

This can be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

URL Rewriting
You can append some extra data at the end of each URL. This data identifies the session; the server can associate that session identifier with the data it has stored about that session.

For example, with http://tutorialspoint.com/file.htm;sessionid=12345, the session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies. The drawback here is that you will have to generate every URL dynamically to assign a session ID though page is a simple static HTML page.

The session Object

Apart from the above mentioned options, JSP makes use of the servlet provided HttpSession Interface. This interface provides a way to identify a user across.

a one page request or
visit to a website or
store information about that user
By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows −

<%@ page session = "false" %>
The JSP engine exposes the HttpSession object to the JSP author through the implicit session object. Since session object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or getSession().

Here is a summary of important methods available through the session object −

S.No.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	**public Enumeration getAttributeNames()** This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.
5	**public long getLastAccessedTime()** This method returns the last time the client sent a request associated with the this session, as the number of milliseconds since midnight January 1, 1970 GMT.
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
7	**public void invalidate()** This method invalidates this session and unbinds any objects bound to it.
8	**public booleanisNew()** This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	**public void removeAttribute(String name)** This method removes the object bound with the specified name from this session.
10	**public void setAttribute(String name, Object value)** This method binds an object to this session, using the name specified.
11	**public void setMaxInactiveInterval(int interval)** This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

Session Tracking Example

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
<%@ page import = "java.io.*,java.util.*" %>
<%
```

```
   // Get session creation time.
   Date createTime = new Date(session.getCreationTime());

   // Get last access time of this Webpage.
   Date lastAccessTime = new Date(session.getLastAccessedTime());

   String title = "Welcome Back to my website";
   Integer visitCount = new Integer(0);
   String visitCountKey = new String("visitCount");
   String userIDKey = new String("userID");
   String userID = new String("ABCD");

   // Check if this is new comer on your Webpage.
   if (session.isNew() ){
      title = "Welcome to my website";
session.setAttribute(userIDKey, userID);
session.setAttribute(visitCountKey,  visitCount);
   }
visitCount = (Integer)session.getAttribute(visitCountKey);
visitCount = visitCount + 1;
userID = (String)session.getAttribute(userIDKey);
session.setAttribute(visitCountKey,  visitCount);
%>

<html>
<head>
<title>Session Tracking</title>
</head>

<body>
<center>
<h1>Session Tracking</h1>
</center>

<table border = "1" align = "center">
<tr bgcolor = "#949494">
<th>Session info</th>
<th>Value</th>
</tr>
<tr>
```

```
<td>id</td>
<td><% out.print( session.getId()); %></td>
</tr>
<tr>
<td>Creation Time</td>
<td><% out.print(createTime); %></td>
</tr>
<tr>
<td>Time of Last Access</td>
<td><% out.print(lastAccessTime); %></td>
</tr>
<tr>
<td>User ID</td>
<td><% out.print(userID); %></td>
</tr>
<tr>
<td>Number of visits</td>
<td><% out.print(visitCount); %></td>
</tr>
</table>

</body>
</html>
```

JSP Cookies:

JSP Cookies – Cookies can defined as a small file that will be stored in a browser, it is utilized information tracing purpose. Already Splessons have discussed the cookies concept in servlet technology Servlet Cookies. Following is the list of important useful methods associated with the Cookie object which you can use while manipulating cookies in JSP.

- public void setDomainStringpattern
- public String getDomain
- public String getName
- public String getValue
- public String getPath
- public String getComment

Following is the example which describes more about the cookies. Following is the code to set the cookies.

index.jsp

```
<html>
<body bgcolor="skyblue">
<form action="main.jsp" method="GET">
<center><imgsrc="E:/splessons.png"></br></br>
First Name: <input type="text" name="first_name"></br>
<br />
Last Name: <input type="text" name="last_name"/></br></br>
<input type="submit" value="Submit" /></center>
</form>
</body>
</html>
```

Here just created two forms they are First Name, Last Name and also created **Submit** button. The **GET**method is the default method to pass data from client to web server and it delivers a long string that shows up in the browser's Location. Never utilize the GET method in the event that you have secret key or other touchy data to go to the server. The GET method has size constraint: just 1024 characters can be in a solicitation string.

main.jsp

```
<%
// Create cookies for first and last names.
Cookie firstName = new Cookie("first_name",
request.getParameter("first_name"));
Cookie lastName = new Cookie("last_name",
request.getParameter("last_name"));
// Set expiry date after 24 Hrs for both the cookies.
firstName.setMaxAge(60*60*24);
lastName.setMaxAge(60*60*24);
// Add both the cookies in the response header.
response.addCookie( firstName );
response.addCookie( lastName );
%>
```

```
<html>
<head>
<title>Setting Cookies</title>
</head>
<body>
<center>
<h1>Setting Cookies</h1>
</center>
<ul>
<li><p><b>First Name:</b>
<%= request.getParameter("first_name")%>
</p></li>
<li><p><b>Last Name:</b>
<%= request.getParameter("last_name")%>
</p></li>
</ul>
</body>
</html>
```

The **request.getParameter()** is used to retrieve the details from the static page that id **HTML** page.

Lecture 12. PHP

1. *Origins of PHP*
2. Overview of PHP
3. General Syntactic Characteristics
4. Primitives, Operations, and Expressions
5. Control Statements
6. Arrays
7. User-Defined Functions
8. Objects in PHP
9. Files and Directories
10. Pattern Matching
11. PHP CGI computing
12. Cookies with PHP
13. Sessions with PHP
14. Time and Date
15. Working with Server Environment
16. Call other program on server side
17. MySql access using PHP
18. Exception Handling
19. Networking with PHP
20. PHP for web system development

1. *Origins of PHP*

- **PHP is an acronym for *Personal Home Page*, or *Hypertext Preprocessor***
- **Developed by Rasmus Lerdorf to track visitors to his Web site**
- **Started in 1994, gone through several versions. Current version is PHP6.**
- **PHP is used for server side form handling, file processing, and database access**
- **PHP is an open-source product**

2. Overview of PHP

- PHP is a server-side scripting language whose scripts are embedded in HTML

- PHP is an alternative to Perl for CGI
  Similar to
  Active Server Pages (ASP)
  Java Server Pages (JSP)

- As programming language, PHP has a lot of similarity to that of Perl
  e.g. a variable starts with $, and dynamic type

- The PHP processor has two modes:
  copy HTML and interpret PHP

1. Browser makes a request to a web server for a PHP document

2. Web server, call the PHP processor and inputs the PHP document to the PHP processor. A traditional CGI approach.

3. When PHP processor processes the PHP document, it copies the html portion to an output HTML/XML document, and interprets the PHP script and executes the scripts and then insert the results to the HTML/XML document, finally output the HTML/XML document

4. When it is done, the HTML document is handed to the Web server and it is then sent to the browser

- PHP script can be executed by call:

  php file_name.php

Run PHP as Apache module

PHP can be installed on Apache as a module. i.e. the loaded PHP interpreter with Apache

Such PHP interpreter runs in part of Apache process, and PHP interpreter functions can be called directly

Gain a bit performance, but less secured than the traditional CGI approach.

- **The example of Hello World in HTML**

```
<html>
 <head><title> Hello World </title>
 </head>
 <body>
 <?PHP
  print(“Hello World!“);   //or print “Hello World!“;
 ?>
 </body>
 </html>
```

3. General Syntactic Characteristics

- PHP code can be specified in an HTML document internally or externally:

Internally:

```
<?php
  //... PHP code here
?>
```

Externally:

```
<?php
  include("myScript.php"); //or
  require("myScript.php");
?>
```

The differences error handling.

The file can have both PHP and HTML. If the file has PHP, the PHP must be in `<?php .. ?>`, even if the `include` is already in `<?php .. ?>`

- A document contain HTML and PHP script should have extension .PHP, PHP3, or .phtml

See example

- *Comments* - three different kinds (Java and Perl)

```
// ...
# ...
/* ... */
```

- PHP statements are terminated with semicolons
- Compound statements are formed with braces
- Variables are case sensitive.
- There are 31 reserved words.

- **Output**

- Output from a PHP script is HTML that is sent to the browser

- HTML is sent to the browser through standard output

There are three ways to produce output:

`echo` **can have one or more parameters**
`print` **similar to echo, but only allow on parameter**
`printf` **is similar to that of C language**

echo and `print` take a string, but will coerce other values to strings

```
echo "whatever";

echo("first <br />", $sum)

print "Welcome to my site!";
```

-

4. Primitives, Operations and Expressions

- *Variables*
  - No type declarations, i.e., dynamic type
  - An unassigned variable has the value, `NULL`
  - The *`unset`* function sets a variable to `NULL`
  - The *`IsSet`* function is used to determine whether a variable is `NULL`

•PHP has predefined variables, including the environment variables. The list of the predefined variables can be obtained by calling *`phpinfo()`*

- **Eight primitive types of variables**
  - **Four scalar types**

    Boolean, integer, double, and string
  - **Two compound types**

    array and object
  - **Two special types**

    resource and `NULL`
- **Boolean**

  values are true and false

  0 and "" and "0" are false;
  others are true

- **Integer and double**

  Arithmetic Operators and Expressions

  – If the result of integer division is not an integer, a double is returned

  – Any integer operation that results in overflow produces a double

  – The modulus operator coerces its operands to integer, if necessary

  – When a double is rounded to an integer, the rounding is always towards zero

- ***Arithmetic functions***

  `floor`, `ceil`, `round`, `abs`, `min`, `max`, `rand`, etc.

- **Strings**

  – **Characters are single bytes**
  – **String literals use single or double quotes similar to that of Perl**
    - ***Single-quoted string literals***
      Embedded variables are not interpolated
      Embedded escape sequences are not recognized

    - **Embedded variables in a double-quoted string are interpolated. If there is a variable name but you don't want it interpolated, use backslash**

    - **Embedded escape sequences are recognized or both single- and double-quoted literal strings, embedded delimiters must be backslashed**

  – ***String Operations and Functions***

    - **The only operator is period, for catenation**
    - **Indexing - $str{3} is the fourth character**

String functions:

`strlen`, `strcmp`, `strpos`, `substr`, strstr, substr_replace, str_replace, explode

`chop` – remove whitespace from the right end

`trim` – remove whitespace from both ends

`ltrim` – remove whitespace from the left end

`strtolower`, `strtoupper`

- *Scalar Type Conversions*
  - *String to numeric*

- If the string contains an e or an E, it is converted to double; otherwise to int
- If the string does not begin with a sign or a digit, zero is used

- Explicit conversions – casts

- e.g., `(int)$total` or `intval($total)` or `settype($total, "integer")`

- The type of a variable can be determined with `gettype` or `is_type`

`gettype($total)` - it may return "unknown"

`is_integer($total)` – a predicate function

5. Control Statements

- **Control Expressions**

  - Relational operators , similar to Perl. Special operators (including `===, identical,` and `!==, no identical`)

  - Boolean operators - same as Perl (two sets, `&&` and `and`, etc.)

- Selection statements
  `if, if-else, elseif`
  `switch` - as in C
  - The switch expression type must be integer, double, or string

- Repititions

  `while` - just like C
  `do-while` - just like C
  `for` - just like C
  `foreach` - just like Perl

`break` - in any `for`, `foreach`, `while`, `do-while`, `switch`

`continue` - in any loop

- Alternative compound delimiters – more readability

```
if(...):
   ...
endif;
```

- HTML can be intermingled with PHP script

```
<?PHP
   $a = 7;
   $b = 7;
   if ($a == $b) {
      $a = 3 * $a;
   ?>
   <br /> At this point, $a and $b are
    equal <br />
   So, we change $a to three times $a
   <?PHP
    }
?>
```

PHP - INTRODUCTION

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures *COMandCORBA*, making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

Common uses of PHP

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- You add, delete, modify elements within your database through PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible –

- Simplicity
- Efficiency
- Security
- Flexibility

- Familiarity

"Hello World" Script in PHP

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

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML *orXHTMLifyou'recutting – edge* you'll have PHP statements like this –

Live Demo

```
<html>

   <head>
      <title>Hello World</title>
   </head>

   <body>
      <?php echo "Hello, World!";?>
   </body>

</html>
```

It will produce following result –

```
Hello, World!
```

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

All PHP code must be included inside one of the three special markup tags ATE are recognised by the PHP Parser.

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

<?    PHP code goes here ?>

<script language = "php"> PHP code goes here </script>
```

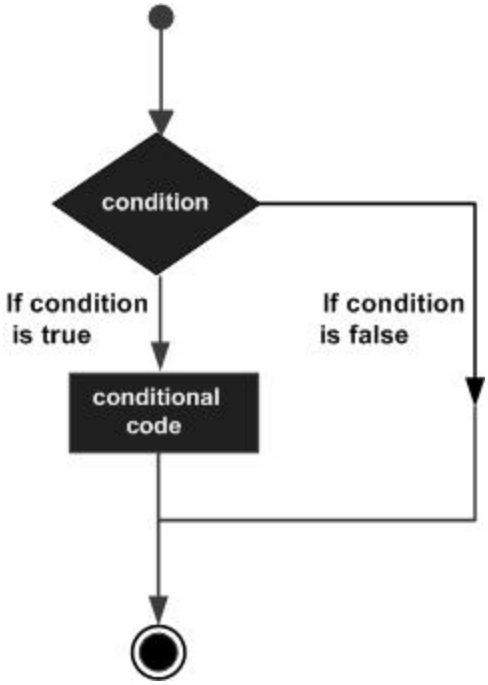
A most common tag is the <?php...?> and we will also use the same tag in our tutorial.

From the next chapter we will start with PHP Environment Setup on your machine and then we will dig out almost all concepts related to PHP to make you comfortable with the PHP language.

PHP - DECISION MAKING

The if, elseif ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements –



- **if...else statement** – use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **elseif statement** – is used with the if...else statement to execute a set of code if **one** of the several condition is true
- **switch statement** – is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

Syntax

```
if (condition)
   code to be executed if condition is true;
else
   code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, Otherwise, it will output "Have a nice day!":

Live Demo

```
<html>
   <body>

      <?php
         $d = date("D");

         if ($d == "Fri")
            echo "Have a nice weekend!";

         else
            echo "Have a nice day!";
      ?>

   </body>
</html>
```

It will produce the following result –

```
Have a nice weekend!
```

The ElseIf Statement

If you want to execute some code if one of the several conditions are true use the elseif statement

Syntax

```
if (condition)
   code to be executed if condition is true;
elseif (condition)
   code to be executed if condition is true;
else
   code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise, it will output "Have a nice day!" –

Live Demo

```
<html>
   <body>

      <?php
         $d = date("D");
```

```
         if ($d == "Fri")
            echo "Have a nice weekend!";

         elseif ($d == "Sun")
            echo "Have a nice Sunday!";

         else
            echo "Have a nice day!";
      ?>

   </body>
</html>
```

It will produce the following result –

```
Have a nice Weekend!
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

```
switch (expression){
   case label1:
      code to be executed if expression = label1;
      break;

   case label2:
      code to be executed if expression = label2;
      break;
      default:

   code to be executed
   if expression is different
   from both label1 and label2;
}
```

Example

The *switch* statement works in an unusual way. First it evaluates given expression then seeks a lable to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the lable matches then statement will execute any specified default code.

Live Demo

```
<html>
   <body>

      <?php
         $d = date("D");
```

```
         switch ($d){
            case "Mon":
               echo "Today is Monday";
               break;

            case "Tue":
               echo "Today is Tuesday";
               break;

            case "Wed":
               echo "Today is Wednesday";
               break;

            case "Thu":
               echo "Today is Thursday";
               break;

            case "Fri":
               echo "Today is Friday";
               break;

            case "Sat":
               echo "Today is Saturday";
               break;

            case "Sun":
               echo "Today is Sunday";
               break;

            default:
               echo "Wonder which day is this ?";
         }
      ?>

   </body>
</html>
```

It will produce the following result –

```
Today is Monday
```

PHP - ARRAYS

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.

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

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

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

Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

Example

Following is the example showing how to create and access numeric arrays.

Here we have used **array** function to create array. This function is explained in function reference.

Live Demo

```
<html>
   <body>

      <?php
         /* First method to create array. */
         $numbers = array( 1, 2, 3, 4, 5);

         foreach( $numbers as $value ) {
            echo "Value is $value <br />";
         }

         /* Second method to create array. */
         $numbers[0] = "one";
         $numbers[1] = "two";
         $numbers[2] = "three";
         $numbers[3] = "four";
         $numbers[4] = "five";

         foreach( $numbers as $value ) {
            echo "Value is $value <br />";
         }
```

```
      ?>

   </body>
</html>
```

This will produce the following result –

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five
```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE – Don't keep associative array inside double quote while printing otherwise it would not return any value.

Example

Live Demo

```
<html>
   <body>

      <?php
         /* First method to associate create array. */
         $salaries = array("mohammad" => 2000, "qadir" => 1000, "zara" => 500);

         echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
         echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
         echo "Salary of zara is ".  $salaries['zara']. "<br />";

         /* Second method to create array. */
         $salaries['mohammad'] = "high";
         $salaries['qadir'] = "medium";
         $salaries['zara'] = "low";

         echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
         echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
         echo "Salary of zara is ".  $salaries['zara']. "<br />";
      ?>
```

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

This will produce the following result –

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low
```

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example

In this example we create a two dimensional array to store marks of three students in three subjects –

This example is an associative array, you can create numeric array in the same fashion.

Live Demo

```
<html>
   <body>

      <?php
         $marks = array(
            "mohammad" => array (
               "physics" => 35,
               "maths" => 30,
               "chemistry" => 39
            ),

            "qadir" => array (
               "physics" => 30,
               "maths" => 32,
               "chemistry" => 29
            ),

            "zara" => array (
               "physics" => 31,
               "maths" => 22,
               "chemistry" => 39
            )
         );

         /* Accessing multi-dimensional array values */
         echo "Marks for mohammad in physics : " ;
         echo $marks['mohammad']['physics'] . "<br />";

         echo "Marks for qadir in maths : ";
         echo $marks['qadir']['maths'] . "<br />";
```

```
         echo "Marks for zara in chemistry : " ;
         echo $marks['zara']['chemistry'] . "<br />";
      ?>

   </body>
</html>
```

This will produce the following result –

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39
```

PHP - LOOP TYPES

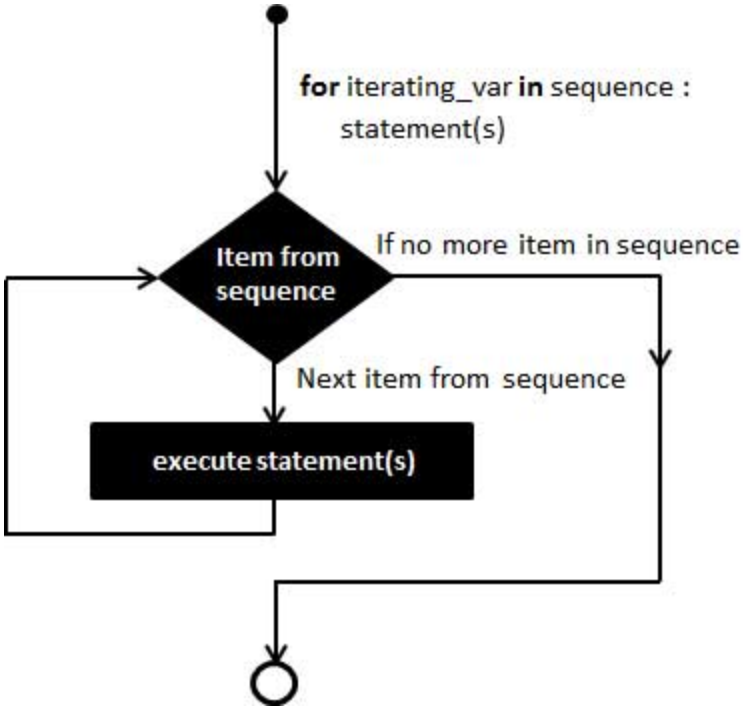
Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for** – loops through a block of code a specified number of times.
- **while** – loops through a block of code if and as long as a specified condition is true.
- **do...while** – loops through a block of code once, and then repeats the loop as long as a special condition is true.
- **foreach** – loops through a block of code for each element in an array.

We will discuss about **continue** and **break** keywords used to control the loops execution.

The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.



Syntax

```
for (initialization; condition; increment){
   code to be executed;
}
```

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop –

Live Demo

```
<html>
   <body>

      <?php
         $a = 0;
         $b = 0;

         for( $i = 0; $i<5; $i++ ) {
            $a += 10;
            $b += 5;
         }

         echo ("At the end of the loop a = $a and b = $b" );
      ?>

   </body>
</html>
```

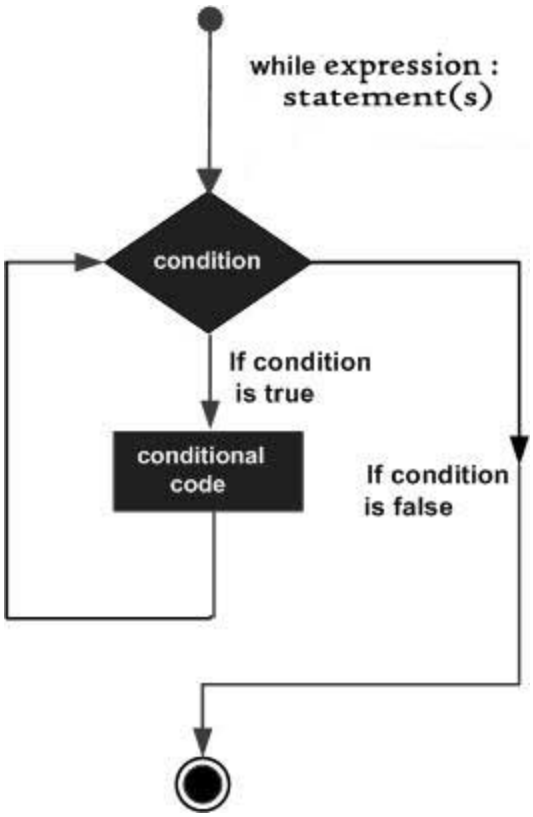
This will produce the following result –

```
At the end of the loop a = 50 and b = 25
```

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.



Syntax

```
while (condition) {
   code to be executed;
}
```

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

Live Demo

```
<html>
   <body>

      <?php
         $i = 0;
         $num = 50;

         while( $i < 10) {
            $num--;
            $i++;
         }

         echo ("Loop stopped at i = $i and num = $num" );
      ?>
```

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

This will produce the following result –

```
Loop stopped at i = 10 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

```
do {
   code to be executed;
}
while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 –

Live Demo

```
<html>
   <body>

      <?php
         $i = 0;
         $num = 0;

         do {
            $i++;
         }

         while( $i < 10 );
         echo ("Loop stopped at i = $i" );
      ?>

   </body>
</html>
```

This will produce the following result –

```
Loop stopped at i = 10
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```
foreach (array as value) {
   code to be executed;
}
```

Example

Try out following example to list out the values of an array.

Live Demo

```
<html>
   <body>

      <?php
         $array = array( 1, 2, 3, 4, 5);

         foreach( $array as $value ) {
            echo "Value is $value <br />";
         }
      ?>

   </body>
</html>
```

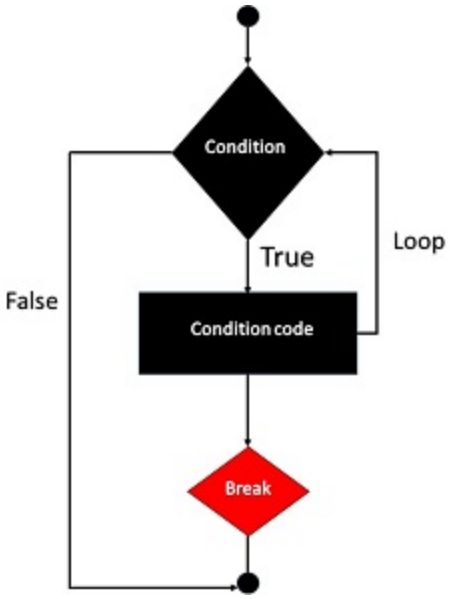
This will produce the following result –

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.



Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

Live Demo

```
<html>
   <body>

      <?php
         $i = 0;

         while( $i < 10) {
            $i++;
            if( $i == 3 )break;
         }
         echo ("Loop stopped at i = $i" );
      ?>

   </body>
</html>
```

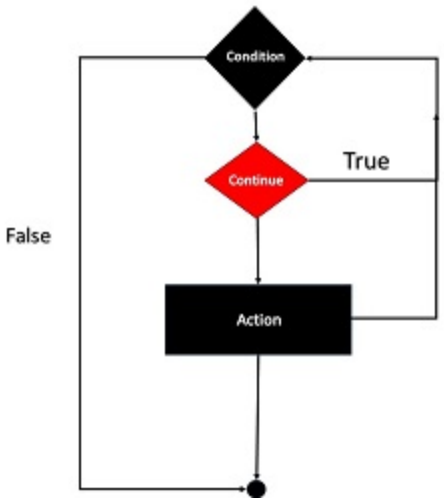
This will produce the following result –

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.



Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

Live Demo

```
<html>
   <body>

      <?php
         $array = array( 1, 2, 3, 4, 5);

         foreach( $array as $value ) {
            if( $value == 3 )continue;
            echo "Value is $value <br />";
         }
      ?>

   </body>
</html>
```

This will produce the following result –

```
Value is 1
Value is 2
Value is 4
Value is 5
```

PHP - FILES & I/O

This chapter will explain following functions related to files –

- Opening a file
- Reading a file
- Writing a file
- Closing a file

Opening and Closing Files

The PHP **fopen** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Sr.No	Mode & Purpose
1	**r** Opens the file for reading only. Places the file pointer at the beginning of the file.
2	**r+** Opens the file for reading and writing. Places the file pointer at the beginning of the file.
3	**w** Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.

4	**w+** Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
5	**a** Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.
6	**a+** Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

If an attempt to open a file fails then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.

After making a changes to the opened file it is important to close it with the **fclose** function. The **fclose** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

Reading a file

Once a file is opened using **fopen** function it can be read with a function called **fread**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The files length can be found using the **filesize** function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

- Open a file using **fopen** function.
- Get the file's length using **filesize** function.

- Read the file's content using **fread** function.
- Close the file with **fclose** function.

The following example assigns the content of a text file to a variable then displays those contents on the web page.

```
<html>

   <head>
      <title>Reading a file using PHP</title>
   </head>

   <body>

      <?php
         $filename = "tmp.txt";
         $file = fopen( $filename, "r" );

         if( $file == false ) {
            echo ( "Error in opening file" );
            exit();
         }

         $filesize = filesize( $filename );
         $filetext = fread( $file, $filesize );
         fclose( $file );

         echo ( "File size : $filesize bytes" );
         echo ( "<pre>$filetext</pre>" );
      ?>

   </body>
</html>
```

It will produce the following result –

```
File size : 278 bytes

The PHP Hypertext Preprocessor (PHP) is a programming
language that allows web developers to create dynamic
content that interacts with databases.
PHP is basically used for developing web based software
applications. This tutorial helps you to build your base
 with PHP.
```

Writing a file

A new file can be written or text can be appended to an existing file using the PHP **fwrite** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would will stop after the specified length has been reached.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using **file_exist** function which takes file name as an argument

```
<?php
   $filename = "/home/user/guest/newfile.txt";
   $file = fopen( $filename, "w" );

   if( $file == false ) {
      echo ( "Error in opening new file" );
      exit();
   }
   fwrite( $file, "This is  a simple test\n" );
   fclose( $file );
?>
<html>

   <head>
      <title>Writing a file using PHP</title>
   </head>

   <body>

      <?php
         $filename = "newfile.txt";
         $file = fopen( $filename, "r" );

         if( $file == false ) {
            echo ( "Error in opening file" );
            exit();
         }

         $filesize = filesize( $filename );
         $filetext = fread( $file, $filesize );

         fclose( $file );

         echo ( "File size : $filesize bytes" );
         echo ( "$filetext" );
         echo("file name: $filename");
      ?>

   </body>
</html>
```

It will produce the following result –

File size : 23 bytes

This is a simple test

file name: newfile.txt

We have covered all the function related to file input and out in PHP File System Function chapter.

PHP - FORM INTRODUCTION

Dynamic Websites

The Websites provide the functionalities that can use to store, update, retrieve, and delete the data in a database.

What is the Form?

A Document that containing black fields, that the user can fill the data or user can select the data.Casually the data will store in the data base

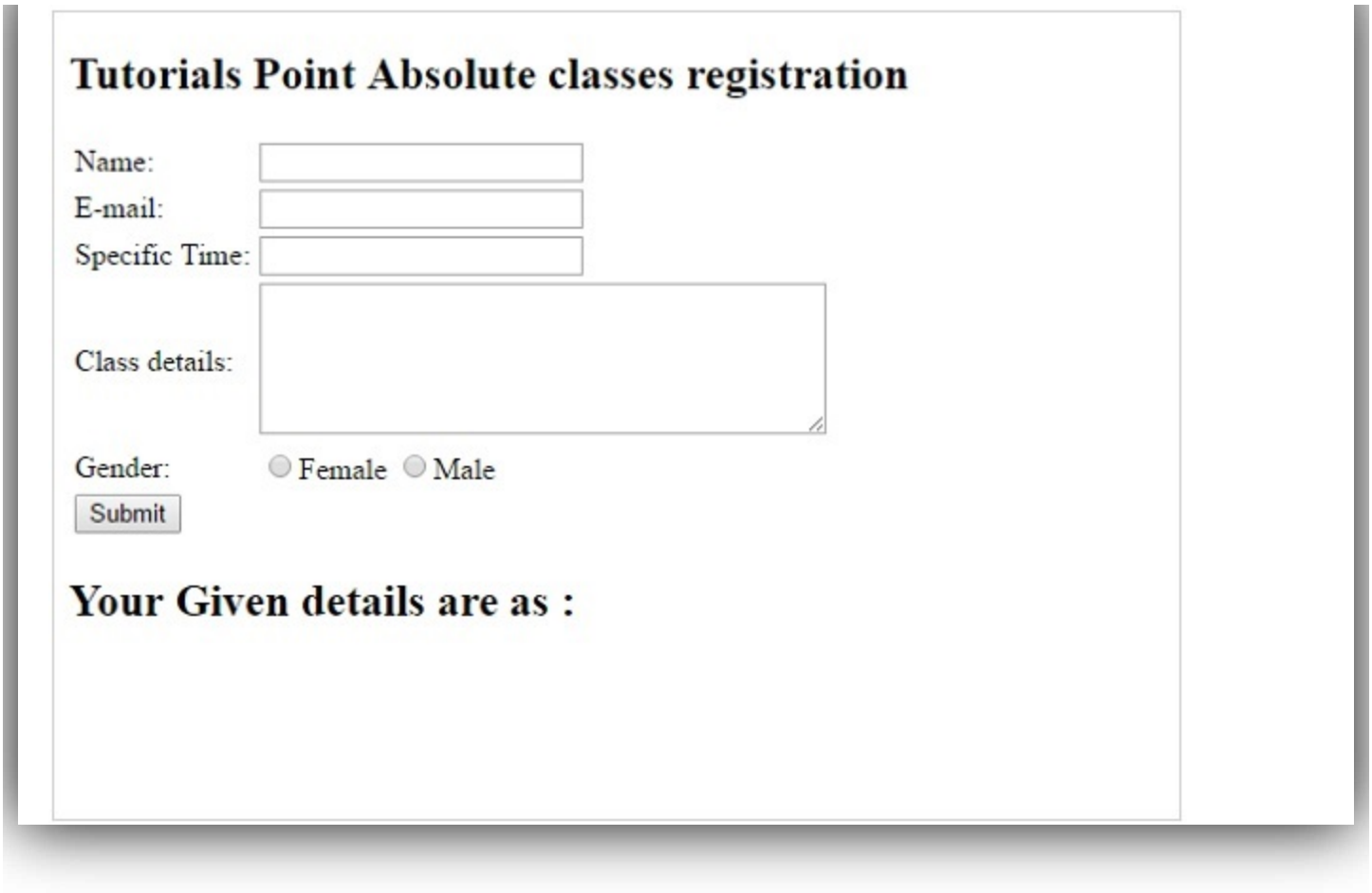
Example

Below example shows the form with some specific actions by using post method.

```
<html>

   <head>
      <title>PHP Form Validation</title>
   </head>

   <body>
      <?php

         // define variables and set to empty values
         $name = $email = $gender = $comment = $website = "";

         if ($_SERVER["REQUEST_METHOD"] == "POST") {
            $name = test_input($_POST["name"]);
            $email = test_input($_POST["email"]);
            $website = test_input($_POST["website"]);
            $comment = test_input($_POST["comment"]);
            $gender = test_input($_POST["gender"]);
         }

         function test_input($data) {
            $data = trim($data);
            $data = stripslashes($data);
            $data = htmlspecialchars($data);
            return $data;
         }
      ?>

      <h2>Tutorials Point Absolute classes registration</h2>

      <form method = "post" action = "">
         <table>
            <tr>
               <td>Name:</td>
               <td><input type = "text" name = "name"></td>
```

```
            </tr>

            <tr>
               <td>E-mail:</td>
               <td><input type = "text" name = "email"></td>
            </tr>

            <tr>
               <td>Specific Time:</td>
               <td><input type = "text" name = "website"></td>
            </tr>

            <tr>
               <td>Class details:</td>
               <td><textarea name = "comment" rows = "5" cols = "40"></textarea></td>
            </tr>

            <tr>
               <td>Gender:</td>
               <td>
                  <input type = "radio" name = "gender" value = "female">Female
                  <input type = "radio" name = "gender" value = "male">Male
               </td>
            </tr>

            <tr>
               <td>
                  <input type = "submit" name = "submit" value = "Submit">
               </td>
            </tr>
         </table>
      </form>

      <?php
         echo "<h2>Your Given details are as :</h2>";
         echo $name;
         echo "<br>";

         echo $email;
         echo "<br>";

         echo $website;
         echo "<br>";

         echo $comment;
         echo "<br>";

         echo $gender;
      ?>

   </body>
</html>
```

It will produce the following result –


Tutorials Point Absolute classes registration
Name:
E-mail:
Specific Time:
Class details:
Gender:
Female
Male
Submit
Your Given details are as :

PHP - COOKIES

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users –

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set cookies, how to access them and how to delete them.

The Anatomy of a Cookie

Cookies are usually set in an HTTP header $although JavaScript can also set a cookie directly on a browser$. A PHP script that sets a cookie might send headers that look something like this –

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
                 path=/; domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server.The browser's headers might look something like this –

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

A PHP script will then have access to the cookie in the environmental variables $_COOKIE or HTTP_COOKIE_VARS[] which holds all cookie names and values. Above cookie can be accessed

using $HTTP_COOKIE_VARS["name"].

Setting Cookies with PHP

PHP provided **setcookie** function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

```
setcookie(name, value, expire, path, domain, security);
```

Here is the detail of all the arguments –

- **Name** – This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** – This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** – This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** – This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** – This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security** – This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies **name** and **age** these cookies will be expired after one hour.

```
<?php
   setcookie("name", "John Watkin", time()+3600, "/","", 0);
   setcookie("age", "36", time()+3600, "/", "",  0);
?>
<html>

   <head>
      <title>Setting Cookies with PHP</title>
   </head>

   <body>
      <?php echo "Set Cookies"?>
   </body>

</html>
```

Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either $_COOKIE or HTTP_COOKIE_VARS variables. Following example will access all the cookies set in above example.

```
<html>
```

```
<head>
   <title>Accessing Cookies with PHP</title>
</head>

<body>

   <?php
      echo $_COOKIE["name"]. "<br />";

      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["name"]. "<br />";

      echo $_COOKIE["age"] . "<br />";

      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["age"] . "<br />";
   ?>

</body>
</html>
```

You can use **isset** function to check if a cookie is set or not.

```
<html>

<head>
   <title>Accessing Cookies with PHP</title>
</head>

<body>

   <?php
      if( isset($_COOKIE["name"]))
         echo "Welcome " . $_COOKIE["name"] . "<br />";

      else
         echo "Sorry... Not recognized" . "<br />";
   ?>

</body>
</html>
```

Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired –

```
<?php
   setcookie( "name", "", time()- 60, "/","", 0);
   setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>
```

```
   <head>
      <title>Deleting Cookies with PHP</title>
   </head>

   <body>
      <?php echo "Deleted Cookies" ?>
   </body>

</html>
```

JDBC - DRIVER TYPES

What is JDBC Driver?

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

The *Java.sql* package that ships with JDK, contains various classes with their behaviours defined and their actual implementaions are done in third-party drivers. Third party vendors implements the *java.sql.Driver* interface in their database driver.

JDBC Drivers Types

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below −

Type 1: JDBC-ODBC Bridge Driver

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name *DSN* that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

The Oracle Call Interface *OCI* driver is an example of a Type 2 driver.

Type 3: JDBC-Net pure Java

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Type 4: 100% Pure Java

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

Which Driver should be Used?

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

PHP - FUNCTIONS

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen** and **fread** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you –

- Creating a PHP Function
- Calling a PHP Function

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Please refer to PHP Function Reference for a complete set of useful functions.

Creating PHP Function

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below –

Live Demo

```
<html>

   <head>
      <title>Writing PHP Function</title>
   </head>

   <body>

      <?php
         /* Defining a PHP Function */
         function writeMessage() {
            echo "You are really a nice person, Have a nice time!";
         }

         /* Calling a PHP Function */
         writeMessage();
      ?>

   </body>
</html>
```

This will display following result –

```
You are really a nice person, Have a nice time!
```

PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

Live Demo

```
<html>

   <head>
      <title>Writing PHP Function with Parameters</title>
   </head>

   <body>

      <?php
         function addFunction($num1, $num2) {
            $sum = $num1 + $num2;
            echo "Sum of the two numbers is : $sum";
         }

         addFunction(10, 20);
      ?>

   </body>
</html>
```

This will display following result –

```
Sum of the two numbers is : 30
```

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Following example depicts both the cases.

Live Demo

```
<html>

   <head>
      <title>Passing Argument by Reference</title>
   </head>

   <body>

      <?php
```

```
         function addFive($num) {
            $num += 5;
         }

         function addSix(&$num) {
            $num += 6;
         }

         $orignum = 10;
         addFive( $orignum );

         echo "Original Value is $orignum<br />";

         addSix( $orignum );
         echo "Original Value is $orignum<br />";
      ?>

   </body>
</html>
```

This will display following result –

```
Original Value is 10
Original Value is 16
```

PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using **return array**1, 2, 3, 4 .

Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

Live Demo

```
<html>

   <head>
      <title>Writing PHP Function which returns value</title>
   </head>

   <body>

      <?php
         function addFunction($num1, $num2) {
            $sum = $num1 + $num2;
            return $sum;
         }
         $return_value = addFunction(10, 20);

         echo "Returned value from the function : $return_value";
      ?>
```

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

This will display following result –

```
Returned value from the function : 30
```

Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it.

Following function prints NULL in case use does not pass any value to this function.

Live Demo

```
<html>

   <head>
      <title>Writing PHP Function which returns value</title>
   </head>

   <body>

      <?php
         function printMe($param = NULL) {
            print $param;
         }

         printMe("This is test");
         printMe();
      ?>

   </body>
</html>
```

This will produce following result –

```
This is test
```

Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Following example depicts this behaviour.

Live Demo

```
<html>

   <head>
      <title>Dynamic Function Calls</title>
   </head>

   <body>

      <?php
         function sayHello() {
```

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

         $function_holder = "sayHello";
         $function_holder();
      ?>

   </body>
</html>
```

This will display following result –

```
Hello
```

PHP - OPERATOR TYPES

What is Operator? Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical *orRelational* Operators
- Assignment Operators
- Conditional *orternary* Operators

Lets have a look on all operators one by one.

Arithmetic Operators

There are following arithmetic operators supported by PHP language –

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

Show Examples

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

Comparison Operators

There are following comparison operators supported by PHP language

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

Show Examples

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

Logical Operators

There are following logical operators supported by PHP language

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

Show Examples

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then condition becomes true.	$AandB$ is true.
or	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	$AorB$ is true.
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	A && B is true.
||	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	$A\|\|B$ is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a	!A && B

	condition is true then Logical NOT operator will make false.	is false.

Assignment Operators

There are following assignment operators supported by PHP language –

Show Examples

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Conditional Operator

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

Show Examples

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

Operators Categories

All the operators we have discussed above can be categorised into following categories –

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

Precedence of PHP Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator –

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.

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

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

PHP - REGULAR EXPRESSIONS

Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.

Using regular expression you can search a particular string inside a another string, you can replace one string by another string and you can split a string into many chunks.

PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. You can use any of them based on your comfort.

- POSIX Regular Expressions
- PERL Style Regular Expressions

POSIX Regular Expressions

The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements *operators* are combined to form more complex expressions.

The simplest regular expression is one that matches a single character, such as g, inside strings such as g, haggle, or bag.

Lets give explanation for few concepts being used in POSIX regular expression. After that we will introduce you with regular expression related functions.

Brackets

Brackets [] have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Sr.No	Expression & Description
1	**[0-9]** It matches any decimal digit from 0 through 9.
2	**[a-z]** It matches any character from lower-case a through lowercase z.
3	**[A-Z]**

	It matches any character from uppercase A through uppercase Z.
4	**[a-Z]** It matches any character from lowercase a through uppercase Z.

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, {int. range}, and $ flags all follow a character sequence.

Sr.No	Expression & Description
1	**p+** It matches any string containing at least one p.
2	**p*** It matches any string containing zero or more p's.
3	**p?** It matches any string containing zero or one p's.
4	**p{N}** It matches any string containing a sequence of **N** p's
5	**p{2,3}** It matches any string containing a sequence of two or three p's.

6	**p{2, }** It matches any string containing a sequence of at least two p's.
7	**p$** It matches any string with p at the end of it.
8	**^p** It matches any string with p at the beginning of it.

Examples

Following examples will clear your concepts about matching characters.

Sr.No	Expression & Description
1	**[^a-zA-Z]** It matches any string not containing any of the characters ranging from a through z and A through Z.
2	**p.p** It matches any string containing p, followed by any character, in turn followed by another p.
3	**^.{2}$** It matches any string containing exactly two characters.
4	**.*** It matches any string enclosed within and .
5	**p***hp**

	It matches any string containing a p followed by zero or more instances of the sequence php.

Predefined Character Ranges

For your programming convenience several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set –

Sr.No	Expression & Description
1	**[[:alpha:]]** It matches any string containing alphabetic characters aA through zZ.
2	**[[:digit:]]** It matches any string containing numerical digits 0 through 9.
3	**[[:alnum:]]** It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
4	**[[:space:]]** It matches any string containing a space.

PHP's Regexp POSIX Functions

PHP currently offers seven functions for searching strings using POSIX-style regular expressions –

Sr.No	Function & Description
1	ereg The ereg function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise.
2	ereg_replace

	The ereg_replace function searches for string specified by pattern and replaces pattern with replacement if found.
3	eregi The eregi function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive.
4	eregi_replace The eregi_replace function operates exactly like ereg_replace, except that the search for pattern in string is not case sensitive.
5	split The split function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
6	spliti The spliti function operates exactly in the same manner as its sibling split, except that it is not case sensitive.
7	sql_regcase The sql_regcase function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

PERL Style Regular Expressions

Perl-style regular expressions are similar to their POSIX counterparts. The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions. In fact, you can use any of the quantifiers introduced in the previous POSIX section.

Lets give explanation for few concepts being used in PERL regular expressions. After that we will introduce you wih regular expression related functions.

Meta characters

A meta character is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for large money sums using the '\d' meta character: **/**[\d]+**ooo/**, Here **\d** will search for any string of numerical character.

Following is the list of meta characters which can be used in PERL Style Regular Expressions.

```
Character               Description
.              a single character
\s             a whitespace character (space, tab, newline)
\S             non-whitespace character
\d             a digit (0-9)
\D             a non-digit
\w             a word character (a-z, A-Z, 0-9, _)
\W             a non-word character
[aeiou]        matches a single character in the given set
[^aeiou]       matches a single character outside the given set
(foo|bar|baz)  matches any of the alternatives specified
```

Modifiers

Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

```
Modifier        Description
i       Makes the match case insensitive
m       Specifies that if the string has newline or carriage
        return characters, the ^ and $ operators will now
        match against a newline boundary, instead of a
        string boundary
o       Evaluates the expression only once
s       Allows use of . to match a newline character
x       Allows you to use white space in the expression for clarity
g       Globally finds all matches
cg      Allows a search to continue even after a global match fails
```

PHP's Regexp PERL Compatible Functions

PHP offers following functions for searching strings using Perl-compatible regular expressions –

Sr.No	Function & Description
1	preg_match The preg_match function searches string for pattern, returning true if pattern exists, and false otherwise.
2	preg_match_all The preg_match_all function matches all occurrences of pattern in string.

3	preg_replace The preg_replace function operates just like ereg_replace, except that regular expressions can be used in the pattern and replacement input parameters.
4	preg_split The preg_split function operates exactly like split, except that regular expressions are accepted as input parameters for pattern.
5	preg_grep The preg_grep function searches all elements of input_array, returning all elements matching the regexp pattern.
6	preg_quote Quote regular expression characters

PHP - SESSIONS

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the **php.ini** file called **session.save_path**. Before using any session variable make sure you have setup this path.

When a session is started following things happen –

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the **session_start** function.This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session_start** at the beginning of the page.

Session variables are stored in associative array called **$_SESSION[]**. These variables can be accessed during lifetime of a session.

The following example starts a session then register a variable called **counter** that is incremented each time the page is visited during the session.

Make use of **isset** function to check if session variable is already set or not.

Put this code in a test.php file and load this file many times to see the result –

Live Demo

```
<?php
   session_start();

   if( isset( $_SESSION['counter'] ) ) {
      $_SESSION['counter'] += 1;
```

```
   }else {
      $_SESSION['counter'] = 1;
   }

   $msg = "You have visited this page ".  $_SESSION['counter'];
   $msg .= "in this session.";
?>

<html>

   <head>
      <title>Setting up a PHP session</title>
   </head>

   <body>
      <?php  echo ( $msg ); ?>
   </body>

</html>
```

It will produce the following result –

```
You have visited this page 1in this session.
```

Destroying a PHP Session

A PHP session can be destroyed by **session_destroy** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use **unset** function to unset a session variable.

Here is the example to unset a single variable –

```
<?php
   unset($_SESSION['counter']);
?>
```

Here is the call which will destroy all the session variables –

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

Turning on Auto Session

You don't need to call start_session function to start a session when a user visits your site if you can set **session.auto_start** variable to 1 in **php.ini** file.

Sessions without cookies

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string.

Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

Live Demo

```
<?php
   session_start();

   if (isset($_SESSION['counter'])) {
      $_SESSION['counter'] = 1;
   }else {
      $_SESSION['counter']++;
   }

   $msg = "You have visited this page ".  $_SESSION['counter'];
   $msg .= "in this session.";

   echo ( $msg );
?>

<p>
   To continue  click following link <br />

   <a  href = "nextpage.php?<?php echo htmlspecialchars(SID); ?>">
</p>
```

It will produce the following result –

```
You have visited this page 1in this session.
To continue click following link
```

The **htmlspecialchars** may be used when printing the SID in order to prevent XSS related attacks.

PHP & MYSQL

PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.

What you should already have ?

- You have gone through MySQL tutorial to understand MySQL Basics.
- Downloaded and installed a latest version of MySQL.
- Created database user **guest** with password **guest123**.
- If you have not created a database then you would need root user and its password to create a database.

We have divided this chapter in the following sections –

- **Connecting to MySQL database** – Learn how to use PHP to open and close a MySQL database connection.
- **Create MySQL Database Using PHP** – This part explains how to create MySQL database and tables using PHP.
- **Delete MySQL Database Using PHP** – This part explains how to delete MySQL database and tables using PHP.
- **Insert Data To MySQL Database** – Once you have created your database and tables then you would like to insert your data into created tables. This session will take you through real example on data insert.
- **Retrieve Data From MySQL Database** – Learn how to fetch records from MySQL database using PHP.
- **Using Paging through PHP** – This one explains how to show your query result into multiple pages and how to create the navigation link.
- **Updating Data Into MySQL Database** – This part explains how to update existing records into MySQL database using PHP.
- **Deleting Data From MySQL Database** – This part explains how to delete or purge existing records from MySQL database using PHP.
- **Using PHP To Backup MySQL Database** – Learn different ways to take backup of your MySQL database for safety purpose.

JDBC - INTRODUCTION

What is JDBC?

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

- Java Applications
- Java Applets
- Java Servlets
- Java ServerPages *JSPs*
- Enterprise JavaBeans *EJBs*.

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

Pre-Requisite

Before moving further, you need to have a good understanding of the following two subjects −

- Core JAVA Programming
- SQL or MySQL Database

JDBC Architecture

The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with

respect to the JDBC drivers and the Java application −

Common JDBC Components

The JDBC API provides the following interfaces and classes −

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

The JDBC 4.0 Packages

The java.sql and javax.sql are the primary packages for JDBC 4.0. This is the latest JDBC version at the time of writing this tutorial. It offers the main classes for interacting with your data sources.

The new features in these packages include changes in the following areas −

- Automatic database driver loading.
- Exception handling improvements.
- Enhanced BLOB/CLOB functionality.
- Connection and statement interface enhancements.
- National character set support.

- SQL ROWID access.
- SQL 2003 XML data type support.
- Annotations

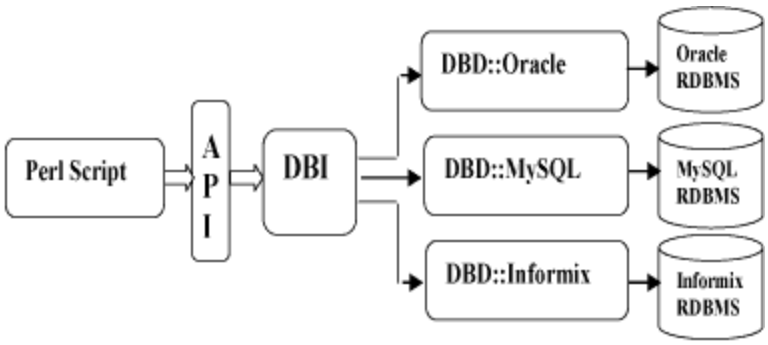
PERL - DATABASE ACCESS

This chapter teaches you how to access a database inside your Perl script. Starting from Perl 5 has become very easy to write database applications using **DBI** module. DBI stands for **Database Independent Interface** for Perl, which means DBI provides an abstraction layer between the Perl code and the underlying database, allowing you to switch database implementations really easily.

The DBI is a database access module for the Perl programming language. It provides a set of methods, variables, and conventions that provide a consistent database interface, independent of the actual database being used.

Architecture of a DBI Application

DBI is independent of any database available in backend. You can use DBI whether you are working with Oracle, MySQL or Informix, etc. This is clear from the following architure diagram.



Here DBI is responsible of taking all SQL commands through the API, *i.e.*, *ApplicationProgrammingInterface* and to dispatch them to the appropriate driver for actual execution. And finally, DBI is responsible of taking results from the driver and giving back it to the calling scritp.

Notation and Conventions

Throughout this chapter following notations will be used and it is recommended that you should also follow the same convention.

```
$dsn    Database source name
$dbh    Database handle object
$sth    Statement handle object
$h      Any of the handle types above ($dbh, $sth, or $drh)
$rc     General Return Code  (boolean: true=ok, false=error)
$rv     General Return Value (typically an integer)
@ary    List of values returned from the database.
$rows   Number of rows processed (if available, else -1)
$fh     A filehandle
undef   NULL values are represented by undefined values in Perl
\%attr  Reference to a hash of attribute values passed to methods
```

Database Connection

Assuming we are going to work with MySQL database. Before connecting to a database make sure of the followings. You can take help of our MySQL tutorial in case you are not aware about how to create database and tables in MySQL database.

- You have created a database with a name TESTDB.
- You have created a table with a name TEST_TABLE in TESTDB.
- This table is having fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- User ID "testuser" and password "test123" are set to access TESTDB.
- Perl Module DBI is installed properly on your machine.
- You have gone through MySQL tutorial to understand MySQL Basics.

Following is the example of connecting with MySQL database "TESTDB" –

```
#!/usr/bin/perl

use DBI
use strict;

my $driver = "mysql";
my $database = "TESTDB";
my $dsn = "DBI:$driver:database=$database";
my $userid = "testuser";
my $password = "test123";

my $dbh = DBI->connect($dsn, $userid, $password ) or die $DBI::errstr;
```

If a connection is established with the datasource then a Database Handle is returned and saved into *dbh for further use otherwise* dbh is set to *undef* value and $DBI::errstr returns an error string.

INSERT Operation

INSERT operation is required when you want to create some records into a table. Here we are using table TEST_TABLE to create our records. So once our database connection is established, we are ready to create records into TEST_TABLE. Following is the procedure to create single record into TEST_TABLE. You can create as many as records you like using the same concept.

Record creation takes the following steps –

- Preparing SQL statement with INSERT statement. This will be done using **prepare** API.
- Executing SQL query to select all the results from the database. This will be done using **execute** API.
- Releasing Stattement handle. This will be done using **finish** API.
- If everything goes fine then **commit** this operation otherwise you can **rollback** complete transaction. Commit and Rollback are explained in next sections.

```
my $sth = $dbh->prepare("INSERT INTO TEST_TABLE
                        (FIRST_NAME, LAST_NAME, SEX, AGE, INCOME )
                          values
```

```
                        ('john', 'poul', 'M', 30, 13000)");
$sth->execute() or die $DBI::errstr;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

Using Bind Values

There may be a case when values to be entered is not given in advance. So you can use bind variables which will take the required values at run time. Perl DBI modules make use of a question mark in place of actual value and then actual values are passed through execute API at the run time. Following is the example –

```
my $first_name = "john";
my $last_name = "poul";
my $sex = "M";
my $income = 13000;
my $age = 30;
my $sth = $dbh->prepare("INSERT INTO TEST_TABLE
                        (FIRST_NAME, LAST_NAME, SEX, AGE, INCOME )
                          values
                        (?,?,?,?,?)");
$sth->execute($first_name,$last_name,$sex, $age, $income)
          or die $DBI::errstr;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

READ Operation

READ Operation on any databasse means to fetch some useful information from the database, i.e., one or more records from one or more tables. So once our database connection is established, we are ready to make a query into this database. Following is the procedure to query all the records having AGE greater than 20. This will take four steps –

- Preparing SQL SELECT query based on required conditions. This will be done using **prepare** API.
- Executing SQL query to select all the results from the database. This will be done using **execute** API.
- Fetching all the results one by one and printing those results.This will be done using **fetchrow_array** API.
- Releasing Stattement handle. This will be done using **finish** API.

```
my $sth = $dbh->prepare("SELECT FIRST_NAME, LAST_NAME
                        FROM TEST_TABLE
                        WHERE AGE > 20");
$sth->execute() or die $DBI::errstr;
print "Number of rows found :" + $sth->rows;
while (my @row = $sth->fetchrow_array()) {
   my ($first_name, $last_name ) = @row;
   print "First Name = $first_name, Last Name = $last_name\n";
}
$sth->finish();
```

Using Bind Values

There may be a case when condition is not given in advance. So you can use bind variables, which will take the required values at run time. Perl DBI modules makes use of a question mark in place of actual value and then the actual values are passed through execute API at the run time. Following is the example –

```
$age = 20;
my $sth = $dbh->prepare("SELECT FIRST_NAME, LAST_NAME
                         FROM TEST_TABLE
                         WHERE AGE > ?");
$sth->execute( $age ) or die $DBI::errstr;
print "Number of rows found :" + $sth->rows;
while (my @row = $sth->fetchrow_array()) {
   my ($first_name, $last_name ) = @row;
   print "First Name = $first_name, Last Name = $last_name\n";
}
$sth->finish();
```

UPDATE Operation

UPDATE Operation on any database means to update one or more records already available in the database tables. Following is the procedure to update all the records having SEX as 'M'. Here we will increase AGE of all the males by one year. This will take three steps –

- Preparing SQL query based on required conditions. This will be done using **prepare** API.
- Executing SQL query to select all the results from the database. This will be done using **execute** API.
- Releasing Stattement handle. This will be done using **finish** API.
- If everything goes fine then **commit** this operation otherwise you can **rollback** complete transaction. See next section for commit and rollback APIs.

```
my $sth = $dbh->prepare("UPDATE TEST_TABLE
                         SET   AGE = AGE + 1
                         WHERE SEX = 'M'");
$sth->execute() or die $DBI::errstr;
print "Number of rows updated :" + $sth->rows;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

Using Bind Values

There may be a case when condition is not given in advance. So you can use bind variables, which will take required values at run time. Perl DBI modules make use of a question mark in place of actual value and then the actual values are passed through execute API at the run time. Following is the example –

```
$sex = 'M';
my $sth = $dbh->prepare("UPDATE TEST_TABLE
                         SET   AGE = AGE + 1
                         WHERE SEX = ?");
$sth->execute('$sex') or die $DBI::errstr;
print "Number of rows updated :" + $sth->rows;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

In some case you would like to set a value, which is not given in advance so you can use binding value as follows. In this example income of all males will be set to 10000.

```
$sex = 'M';
$income = 10000;
my $sth = $dbh->prepare("UPDATE TEST_TABLE
                         SET   INCOME = ?
                         WHERE SEX = ?");
$sth->execute( $income, '$sex') or die $DBI::errstr;
print "Number of rows updated :" + $sth->rows;
$sth->finish();
```

DELETE Operation

DELETE operation is required when you want to delete some records from your database. Following is the procedure to delete all the records from TEST_TABLE where AGE is equal to 30. This operation will take the following steps.

- Preparing SQL query based on required conditions. This will be done using **prepare** API.
- Executing SQL query to delete required records from the database. This will be done using **execute** API.
- Releasing Stattement handle. This will be done using **finish** API.
- If everything goes fine then **commit** this operation otherwise you can **rollback** complete transaction.

```
$age = 30;
my $sth = $dbh->prepare("DELETE FROM TEST_TABLE
                         WHERE AGE = ?");
$sth->execute( $age ) or die $DBI::errstr;
print "Number of rows deleted :" + $sth->rows;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

Using do Statement

If you're doing an UPDATE, INSERT, or DELETE there is no data that comes back from the database, so there is a short cut to perform this operation. You can use **do** statement to execute any of the command as follows.

```
$dbh->do('DELETE FROM TEST_TABLE WHERE age =30');
```

do returns a true value if it succeeded, and a false value if it failed. Actually, if it succeeds it returns the number of affected rows. In the example it would return the number of rows that were actually deleted.

COMMIT Operation

Commit is the operation which gives a green signal to database to finalize the changes and after this operation no change can be reverted to its orignal position.

Here is a simple example to call **commit** API.

```
$dbh->commit or die $dbh->errstr;
```

ROLLBACK Operation

If you are not satisfied with all the changes or you encounter an error in between of any operation , you can revert those changes to use **rollback** API.

Here is a simple example to call **rollback** API.

```
$dbh->rollback or die $dbh->errstr;
```

Begin Transaction

Many databases support transactions. This means that you can make a whole bunch of queries which would modify the databases, but none of the changes are actually made. Then at the end, you issue the special SQL query **COMMIT**, and all the changes are made simultaneously. Alternatively, you can issue the query ROLLBACK, in which case all the changes are thrown away and database remains unchanged.

Perl DBI module provided **begin_work** API, which enables transactions *byturningAutoCommitoff* until the next call to commit or rollback. After the next commit or rollback, AutoCommit will automatically be turned on again.

```
$rc  = $dbh->begin_work  or die $dbh->errstr;
```

AutoCommit Option

If your transactions are simple, you can save yourself the trouble of having to issue a lot of commits. When you make the connect call, you can specify an **AutoCommit** option which will perform an automatic commit operation after every successful query. Here's what it looks like –

```
my $dbh = DBI->connect($dsn, $userid, $password,
              {AutoCommit => 1})
              or die $DBI::errstr;
```

Here AutoCommit can take value 1 or 0, where 1 means AutoCommit is on and 0 means AutoCommit is off.

Automatic Error Handling

When you make the connect call, you can specify a RaiseErrors option that handles errors for you automatically. When an error occurs, DBI will abort your program instead of returning a failure code. If all you want is to abort the program on an error, this can be convenient. Here's what it looks like –

```
my $dbh = DBI->connect($dsn, $userid, $password,
              {RaiseError => 1})
              or die $DBI::errstr;
```

Here RaiseError can take value 1 or 0.

Disconnecting Database

To disconnect Database connection, use **disconnect** API as follows –

```
$rc = $dbh->disconnect  or warn $dbh->errstr;
```

The transaction behaviour of the disconnect method is, sadly, undefined. Some database systems $such as Oracle and Ingres$ will automatically commit any outstanding changes, but others $such as Informix$ will rollback any outstanding changes. Applications not using AutoCommit should explicitly call commit or rollback before calling disconnect.

Using NULL Values

Undefined values, or undef, are used to indicate NULL values. You can insert and update columns with a NULL value as you would a non-NULL value. These examples insert and update the column age with a NULL value –

```
$sth = $dbh->prepare(qq {
         INSERT INTO TEST_TABLE (FIRST_NAME, AGE) VALUES (?, ?)
       });
$sth->execute("Joe", undef);
```

Here **qq{}** is used to return a quoted string to **prepare** API. However, care must be taken when trying to use NULL values in a WHERE clause. Consider –

```
SELECT FIRST_NAME FROM TEST_TABLE WHERE age = ?
```

Binding an undef $NULL$ to the placeholder will not select rows, which have a NULL age! At least for database engines that conform to the SQL standard. Refer to the SQL manual for your database engine or any SQL book for the reasons for this. To explicitly select NULLs you have to say "WHERE age IS NULL".

A common issue is to have a code fragment handle a value that could be either defined or undef $non - NULL or NULL$ at runtime. A simple technique is to prepare the appropriate statement as needed, and substitute the placeholder for non-NULL cases –

```
$sql_clause = defined $age? "age = ?" : "age IS NULL";
$sth = $dbh->prepare(qq {
         SELECT FIRST_NAME FROM TEST_TABLE WHERE $sql_clause
       });
$sth->execute(defined $age ? $age : ());
```

Some Other DBI Functions

available_drivers

```
@ary = DBI->available_drivers;
@ary = DBI->available_drivers($quiet);
```

Returns a list of all available drivers by searching for DBD::* modules through the directories in @INC. By default, a warning is given if some drivers are hidden by others of the same name in earlier directories. Passing a true value for $quiet will inhibit the warning.

installed_drivers

```
%drivers = DBI->installed_drivers();
```

Returns a list of driver name and driver handle pairs for all drivers 'installed' *loaded* into the current process. The driver name does not include the 'DBD::' prefix.

data_sources

```
@ary = DBI->data_sources($driver);
```

Returns a list of data sources *databases* available via the named driver. If $driver is empty or undef, then the value of the DBI_DRIVER environment variable is used.

quote

```
$sql = $dbh->quote($value);
$sql = $dbh->quote($value, $data_type);
```

Quote a string literal for use as a literal value in an SQL statement, by escaping any special characters *suchasquotationmarks* contained within the string and adding the required type of outer quotation marks.

```
$sql = sprintf "SELECT foo FROM bar WHERE baz = %s",
               $dbh->quote("Don't");
```

For most database types, quote would return 'Don''t' *includingtheouterquotationmarks*. It is valid for the quote method to return an SQL expression that evaluates to the desired string. For example –

```
$quoted = $dbh->quote("one\ntwo\0three")

may produce results which will be equivalent to

CONCAT('one', CHAR(12), 'two', CHAR(0), 'three')
```

Methods Common to All Handles

err

```
$rv = $h->err;
or
$rv = $DBI::err
or
$rv = $h->err
```

Returns the native database engine error code from the last driver method called. The code is typically an integer but you should not assume that. This is equivalent to *DBI* :: *error*h->err.

errstr

```
$str = $h->errstr;
or
$str = $DBI::errstr
or
$str = $h->errstr
```

Returns the native database engine error message from the last DBI method called. This has the same lifespan issues as the "err" method described above. This is equivalent to *DBI* :: *errstror*h->errstr.

rows

```
$rv = $h->rows;
or
$rv = $DBI::rows
```

This returns the number of rows effected by previous SQL statement and equivalent to $DBI::rows.

trace

```
$h->trace($trace_settings);
```

DBI sports an extremely useful ability to generate runtime tracing information of what it's doing, which can be a huge time-saver when trying to track down strange problems in your DBI programs. You can use different values to set trace level. These values varies from 0 to 4. The value 0 means disable trace and 4 means generate complete trace.

Interpolated Statements are Prohibited

It is highly recommended not to use interpolated statements as follows –

```
while ($first_name = <>) {
   my $sth = $dbh->prepare("SELECT *
                            FROM TEST_TABLE
                            WHERE FIRST_NAME = '$first_name'");
   $sth->execute();
   # and so on ...
}
```

Thus don't use interpolated statement instead use **bind value** to prepare dynamic SQL statement.