GraphicEra
Deemed to be University
Accredited by NAAC with Grade A
NBA Accredited Programs in ECE, CSE & ME
Approved by AICTE, Ministry of HRD, Govt. of India
DEHRADUN, UTTARAKHAND, INDIA

WEB AND INTERNET TECHNOLOGIES UNIT I

Introduction to Web Technologies: Introduction to Web servers like Apache 1.1, IIS XAMPP(Bundle Server), WAMP(Bundle Server),Handling HTTP Request and Response, installations of above servers, HTML and CSS: HTML 5.0 , XHTML, CSS 3. **UNIT II**
Java Script: An introduction to JavaScript–JavaScript DOM Model-Date and Objects,- Regular Expressions- Exception Handling-Validation-Built-in objects-Event Handling- DHTML with JavaScript. Servlets: Java Servlet Architecture- Servlet Life Cycle- Form GET and POST actions- Session Handling- Understanding Cookies. Installing and Configuring Apache Tomcat Web Server;- DATABASE CONNECTIVITY: JDBC perspectives, JDBC program example - JSP: Understanding Java Server Pages-JSP Standard Tag Library(JSTL)-Creating HTML forms by embedding JSP code.
UNIT III
Introduction to PHP: The problem with other Technologies (Servelets and JSP), Downloading, installing, configuring PHP, Programming in a Web environment and The anatomy of a PHP Page. Overview of PHP Data types and Concepts: Variables and data types, Operators, Expressions and Statements, Strings, Arrays and Functions. PHP Advanced Concepts: Using Cookies, Using HTTP Headers, Using Sessions,
Authenticating users, Using Environment and Configuration variables, Working with Date and Time.
UNIT IV
Creating and Using Forms: Understanding Common Form Issues, GET vs. POST, Validating form input, Working with multiple forms, and Preventing Multiple Submissions ofa form. XML: Basic XML- Document Type Definition XML Schema DOM and Presenting XML, XML Parsers and Validation, XSL and XSLT Transformation, News Feed (RSS and ATOM).

UNIT V

AJAX: Ajax Client Server Architecture-XML Http Request Object-Call Back Methods; Web Services: Introduction- Java web services Basics – Creating, Publishing, Testing and Describing a Web services (WSDL)-Consuming a web service, Database Driven web service from an application – SOAP.

TEXT BOOKS:

1. Beginning PHP and MySQL, 3rd Edition , Jason Gilmore, Apress Publications (Dream tech.).
2. PHP 5 Recipes A problem Solution Approach Lee Babin, Nathan A Good, Frank M.Kromann and Jon Stephens.
3. Deitel and Deitel and Nieto, ―Internet and World Wide Web - How to Program‖, Prentice Hall, 5 th Edition, 2011.
4. Herbert Schildt, ―Java-The Complete Reference‖, Eighth Edition, Mc Graw Hill Professional, 2011.

Unit-1

Introduction to Web Technologies

* *

Introduction to Web Technologies: Introduction to Web servers like Apache 1.1, IIS XAMPP (Bundle Server), WAMP(Bundle Server),Handling HTTP Request and Response, installations of above servers, HTML and CSS: HTML 5.0 , XHTML, CSS 3.

* *

1.1 Working of Internet

Although the physical network connections, the hardware communication devices and the software communication protocols are required for communication across the Internet, the application software provides useful functionalities.

In a network application, two application programs participate in any communication: one application initiates communication and the other accepts it. This is known as the Client-Server interaction. This is the methodology used for internet communication.

1.1.1 Client-Server

Client and Server are two applications involved in communication. These components work together over a network. It involves the client requesting serve from the server. The Server provides the requested service.

The typical features of the Client are:

- It is front-end of an application.
- It manages user-interface portion.
- It validates data entered by the user.
- It dispatches requests to server program.

The typical features of the Server are:

- Performs a back-end task.
- Receives requests from clients.
- Executes database retrievals and updates.
- Manages data integrity.
- Dispatches response to clients.

Web Browsers

A Web browser is a software program that is used to access the World Wide Web(WWW). It allows users to view Web pages and navigate between them.

Examples of Web Browsers are: Mozilla, Microsoft Internet Explorer, Opera, Chrome, Netscape etc.

Web Browsers are known as Universal Clients because they act as the common Client for all Web-based applications. They are the Web Clients that request services from a Web Server, Which is located some where on the Internet or Intranet.

Server Program & Server System

Genrally, the term `Server' refers to a program that waits for a request and provides service. However, a Computer that runs many such Server programs is also known as a Server.

Computers that have fast CPUs, large memories and powerful operating systems are also called Server Machines(or Server Systems or Server Computers).

—A Server is the program that provides Service to a client".

Working of Server

A server offers one or more Services to clients. By default, it does not do any processing until a client sends in a request. It waits for a client to make a request. This is known as `listening mode of the server.

A typical client server interaction happens as follows:

1. The client sends a request for a server.
2. On receiving a request, the service assigns one of the threads in the pool to process the task and continues to wait for further request.
3. The thread executes the code for the requested service.
4. After execution, it sends the response back to the client.
5. It then returns to the thread pool.

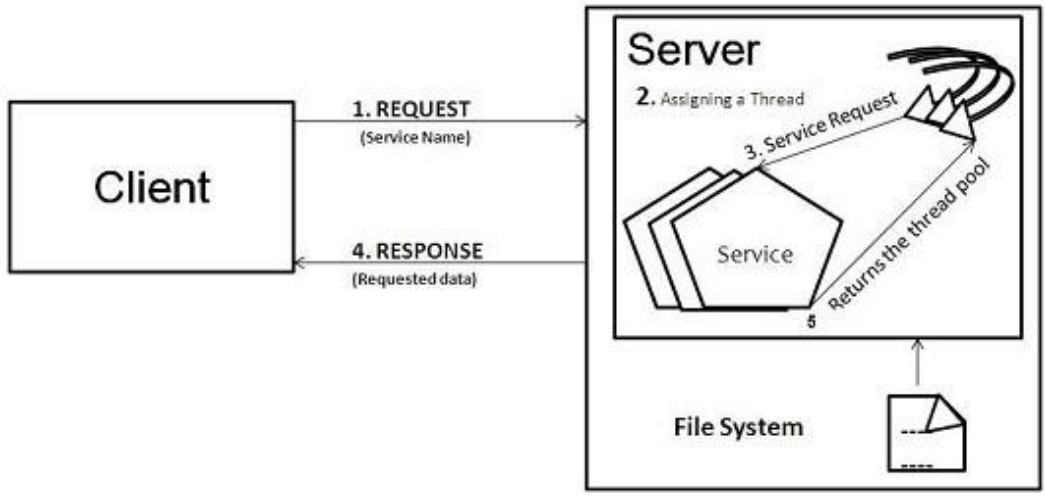


Figure 1.1: Working of Server

1.2 World Wide Web(WWW)

The World Wide Web(WWW) is an information sharing model that allows accessing information over the medium of the Internet. It is the collection of electronic documents that are linked together. These electronic documents are known as `Web Pages'. A collection of related WebPages is known as a `Web Site.

A Web Site is resides on Server computers that are located in around the world. Information on the WWW is always accessible, from anywhere in the world.

The basic architecture is characterized by a Web Browser that displays information content and Web Server that transfer's information to the client. This architecture depends on three key standards for creating, publishing and finding Web documents on the Web:

HTML: Hyper Text Markup Language for creating and editing document content.

URL: Uniform Resource Locator for locating resource on the Internet. **HTTP:**

Hyper Text Transfer Protocol to transfer the data.

1.2.1 HTML: Hyper Text Markup Language

HTML is the authoring language used to create documents on the WWW. HTML makes documents readable across variety of computing plotforms.

1.2.2 URL: Uniform Resource Locator

URL is the unique address that identifies each web page or a resource on the Internet. It indicates where the web pages are stored on the Internet. URL is the standard way of addressing resources on the Internet that are part of WWW.

It supplies the Internet Address of a resource on the WWW, alone with protocol by which the resource is accessed. URLs are used by Web Browsers to connect to a specific server and to get a specific document or page on the Web.

The URL looks like

Protocol://ServerDomainName/Path

Examples

http://www.google.com

Protocol Resource

http://192.168.10.1/download

Protocol IP address of the Resource

1.2.3 HTTP: Hyper Text Transfer Protocol

Web browsers and Web Servers communicate with each other using the HTTP. It is a simple protocol, which standardizes the way requests are sent and processed. This allows different Clients to communicate with any vendor‘s server without compatibility problems.

HTTP is an application level protocol of the TCP/IP suite, which is used to deliver virtually all files and other data on WWW.

It is used to transmit resources that are identified by URL. The most common kinds of resources can be a file, but it can also be dynamically generated content, which is the result of execution of a script or an application on the server.

Features of the HTTP protocol:

- Simple request-response model based protocol.
- Application layer protocol built on TCP/IP.
- Plain-text protocol (Non-Secure)

- Stateless protocol
- Does not define how network connection is initiated or managed Standardized

HTTP Request-Response

HTTP is a simple Request-Response protocol. A HTTP Client, such as a Web Browser initiates a request by establishing a TCP/IP connection to a particular port on a remote host. A HTTPServer listening on that port waits for the Client to send a request, upon receiving the request, the server send back a response.

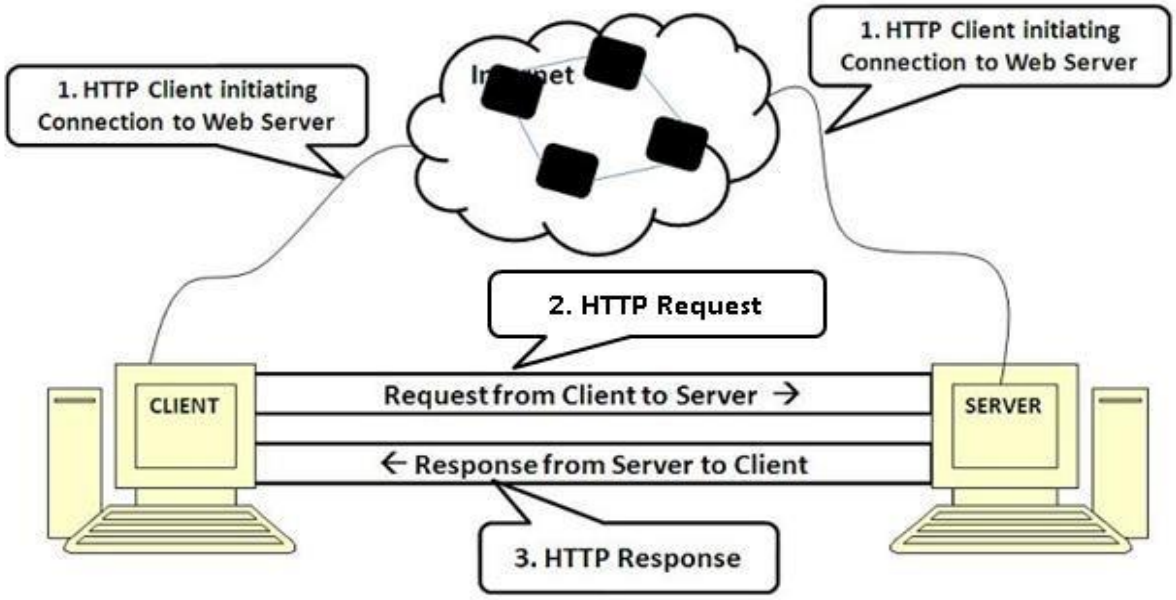


Figure 1.2: HTTP Request-Response

1. A HTTP Client initiates a connection to the Web Server.

2. Once the connection is established, it sends a Request message to the Server.
3. To this message, the server returns a response.

HTTP Request

The HTTP Request has the following message format for transferring entities: A **request line**,Zero or more **header lines**, A**blank line** which separates the headers from the message body.

The request line of the HTTP request includes:

- The method to be applied on the resource.
- The identifier of the resource. The Protocol version in use.

The method filled in request line of HTTP indicates the method to be performed on the object identified by the URL. Some methods are: GET, POST and HEAD.

GET: The GET method is most frequently used method. It is used by default to GET static content. The method can also be used to submit data from a HTML Web Page to the Server. In GET method, the data submitted will be sent as a part of the URL. Hence, in GET method:

- Parameters are encoded and passed along with the URL.
- Usually, parameters are passed as name-value pair.
- There is a physical restriction on the size of it being sent.

POST: A POST method is used to send data as a part of the HTTP message body. In Certain cases the Client may need to send megabytes of information. In these situations POSTmethod is the right choice.

A POST request passes all its data of unlimited length, directly as a part of its HTTPrequest body. The exchange is invisible to client. The URL does not contain the data submitted. Consequently, POST requests cannot be book marked or emailed or in some cases, even reloaded.

Hence, confidential information sent to the Server, such as the credit card number, should be sent via post method.

HEAD: The HEAD method is similar to GET method, except that it asks the server to return only the Response headers and not the content. This method is useful for client to check the characteristics of the resource without actually downloading it, thus saving bandwidth. HTTP Clients usually use the HEAD method when they do not need the files contents.

HEAD is used for the following purposes:

- To determine the document's size.
- To know the document's modification time.
- To know general availability of a Web Page.

HTTP Response

In response to a HTTP Request sent by a HTTP Client, the server sends a HTTP Response.The HTTP Response to requests is usually a program output and not a static file. The first line of a Response message is a **status line**. It consists of

- The protocol version
- Numeric status code
- Description of the status code

HTTP status code: The response status line contains the status of processing of the HTTPrequest. In case of success, it will contain the status code 200 and description ―OK". The status line in this case will be: In case of error, the server sends an appropriate error code back to the Client. The HTTP error codes are standardized. Some of the commonly found error codes: The error or success codes of the HTTP response are standardized in the following manner:

HTTP/1.0 404 Page Not Found

HTTP/1.0 500 Internal Server Error

1XX	Indicates informational message only
2XX	Indicates success of some kinds.
3XX	Redirects the Client to another URL.
4XX	Indicates an error on Client's port.
5XX	Indicates an error on Server's port

HTTP Response Headers: The Response must contain header line describing the following.

MIME-type of the data being sent in response.

- Date and Time stamp.
- Content size etc.

The HTTP Response message body contains the required data.

1.3 Web Servers

A Web Server is a server program running on a computer whose purpose is to serve Web Pages to other computer when required. Every computer on the Internet that contains a Web site will have a Web Server program.

The most common use of web servers is to **host** websites, but there are other uses such as gaming, data storage, running enterprise applications, handling email, FTP, or other web uses.

Many generic web servers also support server-side scripting using Active Server Pages (ASP), PHP, or other scripting languages. This means that the behaviour of the web server can be scripted in separate files, while the actual server **software** remains unchanged. Usually, this function is used to create HTML documents dynamically ("on-the-fly") as opposed to returning static documents.

Examples of Web Servers:

1. Apache Web Server
2. Microsoft Internet Information Server (IIS)
3. XAMPP (Bundle server)
4. WAMP (Bundle server)

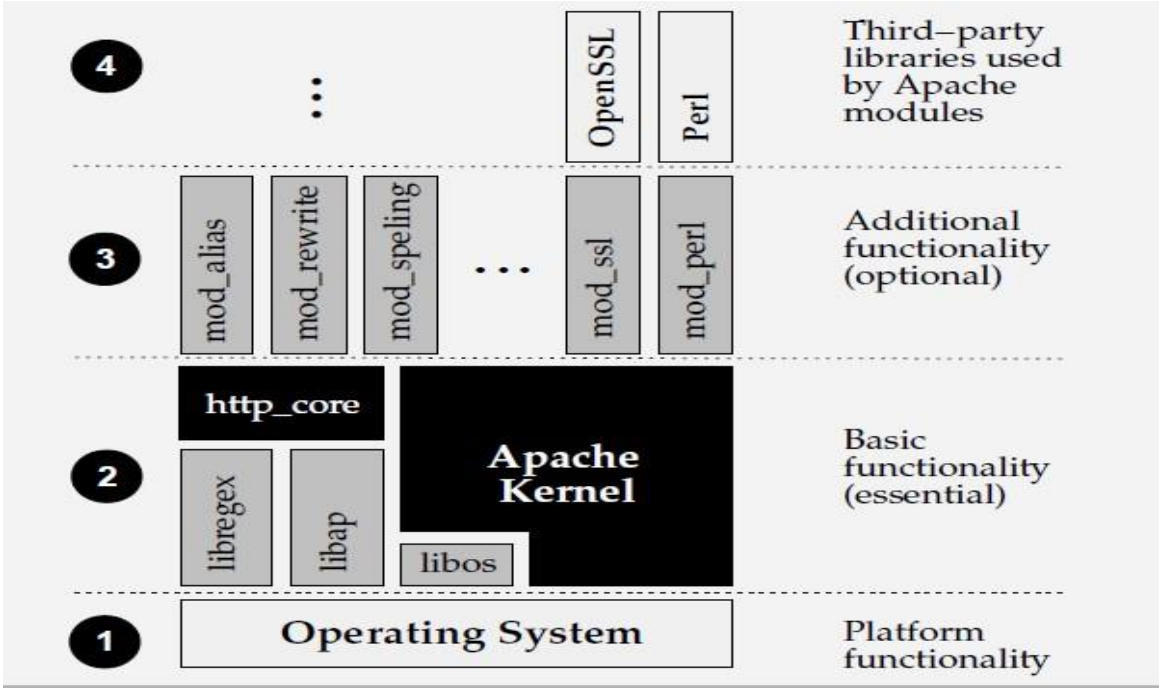
1.3.1 Apache HTTP Server

The Apache HTTP Server commonly referred to as Apache, is a web server program not able for playing a key role in the initial growth of the World Wide Web (WWW). It became the first web server software to exceed the 100 million web site mile stone. Typically Apache is run on a Unix-like Operating system, and was developed for use on Linux.

Apache is developed and maintained by an Open community of developers under the support and approval of the Apache Software Foundation (ASF). The application is available for wide variety of operating system, including UNIX, Free BDS, Solaris, Linux, Novel Netware, OSX, Microsoft Windows, OS/2 etc., Released under the Apache license, Apache is open-source software.

The main design goal of Apache is not to be the fastest Web server, Apache does have performance similar to other \high-performance" Web Servers. Instead of implementing a single architecture Apache provide a variety of Multi Processing Modules (MPMs) which allow Apache to run process-based, where compromises in performance need to be made, and the design of Apache is to reduce latency and increase throughput, relative to simply handling more requests, thusensuring consistent and reliable processing or requests within reasonable time frames. **Features of Apache:**

- It implemented as compiled modules which extend the core functionality, thus the range from server-side programming support to authentication scheme.
- Password-protected pages for a multitude of users (It supports password authentication and digital certificate authentication).
- Customized error pages.
- Display of code in numerous levels of HTML, and the capability to determine at what level the browser can accept the content.
- Virtual hosting allows one Apache installation to serve many different actual Websites. Usage and error logs in multiple and customizable formats Directory Index directives to multiple files.
- URL aliasing or rewriting with no fixed limit
- Robustness and security



Architecture of the Apache Web Server

1.3.2 Microsoft Internet Information Server (IIS)

It is the second most popular Web Server software. It consists of Services including File Transfer Protocol (FTP), Hyper Text Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP) and others that enable a Windows machine to manage Websites. The latest version (IIS 7.6) also includes various modules for security, logging compression and diagnostics.

Because of IIS is provided for Windows systems only, the choice to use IIS necessitates the choice of a Windows Server and therefore increases running costs. Windows is also prone to more malware attacks, and has a reputation as a less secure server option.

But this presents a problem if you'd like to develop and even deploy your PHP-driven website on a Windows server running Microsoft's IIS web server. In recent years, Microsoft, in collaboration with Zend Technologies Ltd., has made great strides towards boosting both the stability and performance of PHP running on both Windows and IIS.

Features:

- IIS has a modular architecture. Modules, also called extensions, can be added or removed individually so that only modules required for specific functionality have to be installed.
- Security Module: Used to perform many tasks related to security in the requestingprocessing pipeline (Authentication Scheme, URL authentication)
- Content Module: Used to perform tasks related to content in the requesting-processing pipeline (Such as processing requests for static pages, returning default page etc.,)
- Compression Module: Used to perform tasks related to compression in the requestingprocessing pipeline (Such as compression responses, performing pre-compression of staticcontent.)
- Caching Module: Used to perform tasks related to caching in the requesting-processing pipeline (Such as storing processed information in the memory on the server and using cached content in subsequent request for the same resource.)
- Logging and Diagnostics Module: Used to perform tasks related to Logging and Diagnostics in the requesting-processing pipeline (Such as passing information and processing status to HTTP.sys for logging, reporting events, and tracking requests currently executing in worker processes.)
- IIS 7.5 includes additional security features: Client-certificate mapping, IP security, Request filtering, URL authentication.

1.3.3 XAMPP (Bundle Server)

XAMPP is a free and open-source cross platform Web Server Solution stack package, consisting mainly of Apache HTTP Server, MySQL database, and interpreter for scripts written in the PHP and Perl programming languages.

X: Cross-Platform
A: Apache
M: MySQL
P: PHP
P: Perl

Officially, XAMPP's designers intended it for use only as a development tool, to allow Website designers and programmers to test their work on their own computer without any access to the Internet. To make this as easy as possible many important security features are disabled by default. XAMPP sometimes used to actually Server Web Pages on the World Wide Web.

Note: XAMPP is also provided support for creating and manipulating databases in MySQL and SQL Lite among others.

Benefits:

- Self contained, multiple instances of XAMPP can exist on a single computer, and any given instance can be copied from one computer to another.
- It automatically starts at system logon.
- You can start and stop Web Server and database stack with one command.

- Run in back ground.
- XAMPP is portable so you can carry it around on a thumb drive.
- The security settings are strict by default, nobody but you will be able to access the WebServer.
- PHP error reporting is enabled by default, which helps when debugging scripts.

1.3.4 WAMP (Bundle Server)

WAMP is the bundle of Apache, MySQL and PHP for Windows. These are the things you need to run a dynamic web sites on your computer in Windows. i.e equal to XAMPP.

WAMPs are packages of independently-created programs installed on computers that use a Microsoft Windows operating system.

Benefits:

- Scripting language that can manipulate the information held in database and generate Web pages dynamically each time the content is requested by browser.
- Other packages are also included like phpMyAdmin which provides a GUI for database manager.

Some of the bundle servers are:

LAMP:Linux, **A**pache, **M**ysql, **P**HP.

SAMP:Solaris, **A**pache, **M**ysql, **P**HP.

MAMP:Mac OS, **A**pache, **M**ysql, **P**HP.

1.4 Installation of Web Servers

1.4.1 Installing Apache and PHP on Windows:

Apache needs to be installed and operational before PHP and MySQL

1. Download the Apache 2.x Win32 MSI installer binary. It's downloadable fromhttp://httpd.apache.org/.

Select the ―Download from a mirror‖ link on the left side of the page and download the best available version. A mirror is a download location. The file that you save to your desktop will be named similarly to apache2.2.4-win32-x86-nossl.msi (the exact version number will vary).

2. Install Apache using the Installation Wizard. Double-click the MSI installer file on your desktop, and you see the installer shown in Figure 1.3

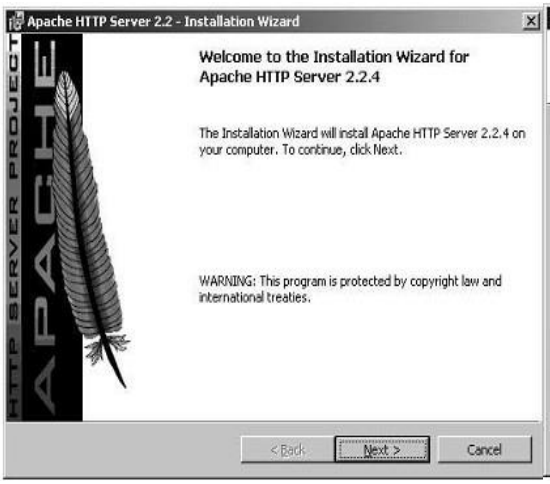


Figure 1.3

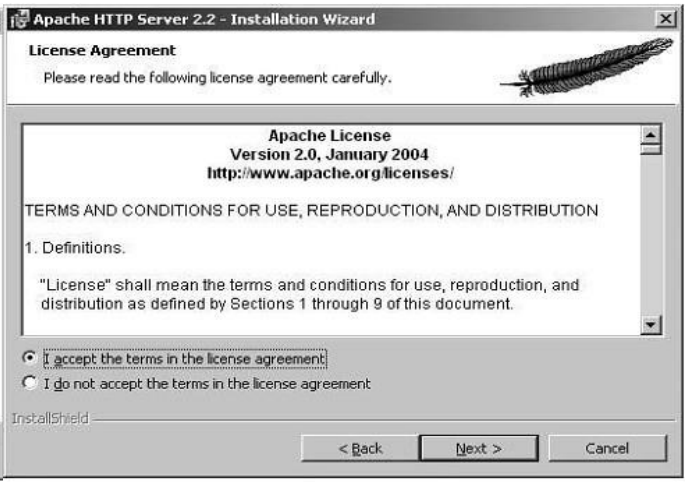


Figure 1.4

3. Accept the license terms by clicking the radio button shown in Figure 1.4. Click Next.

4. You'll see a Read This First box, as shown in Figure 1.5. Additionally, this window offersa number of excellent resources related to the web server. Click Next.

Figure 1.5

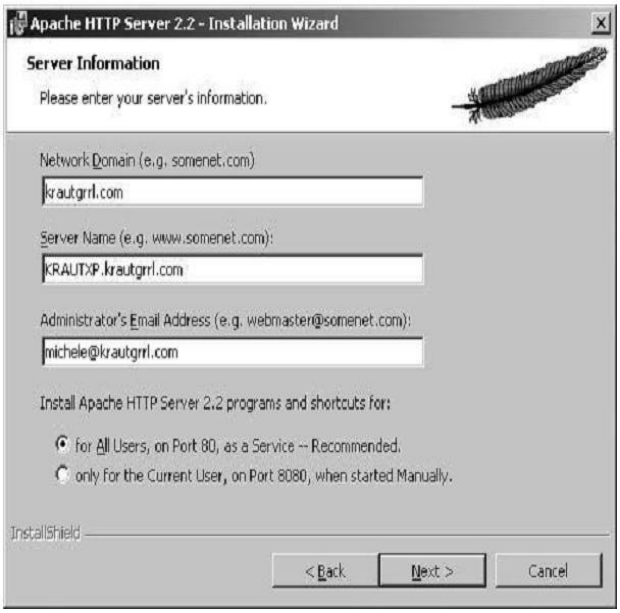


Figure 1.6

5. In the dialog shown in Figure 1.6, enter all pertinent network information. Click Next.
6. In the next screen, shown in Figure 1.7, select the setup type. The Typical install will work for your purposes. Click Next.

Figure 1.7

Figure 1.8

7. Accept the default installation directory, as shown in Figure 1.8. Click Next.
8. As Figure 1.9 shows, it's time to begin the installation. Click Install. The installer installs a variety of modules, and you will see some DOS windows appear and disappear.

Figure 1.9

Figure 1.10

9. Click Finish when the installer is done.
10. Test your installation by entering **http://localhost/** in your browser's location field. Remember, local host is just the name that translates to the IP address **127.0.0.1**, which is always the address of the local computer.
11. After entering the URL in your browser, the default Apache page displays, which is similar to the one shown in Figure 1.10. The installation was successful if you see the text —It works!".This page may be different depending on which version of Apache you install.

    Generally, if you see text that doesn't mention an error, the installation was successful.

Installing PHP

Go to **http://www.php.net/downloads.php** to download the latest version of PHP; both binaries and source code can be found on this web site.

1. The file that you save to your desktop will be named similarly to **php-5.2.1-win32-installer.msi** (the exact version number will vary).
2. Install PHP using the Installation Wizard. Double-click the MSI installer file on your desktop, and you'll see the installer shown in Figure 1.11.

Figure 1.11

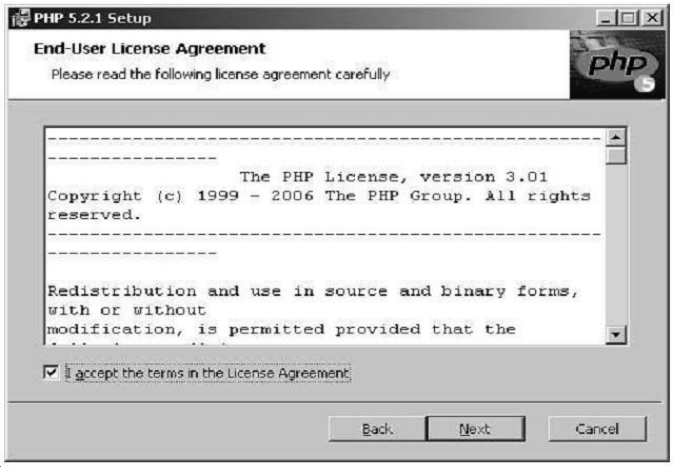


Figure 1.12

3. Click Next. The License Terms dialog appears as shown in Figure 1.12
4. Click the checkbox to accept the licensing terms. Click Next.
5. The Destination Folder dialog appears (see Figure 1.13). Select the destination folder.

You may use the default of **C:\Program Files\PHP** or **C:\PHP** (used in this material that modify the PHP configuration files assume C:\PHP). Click Next.

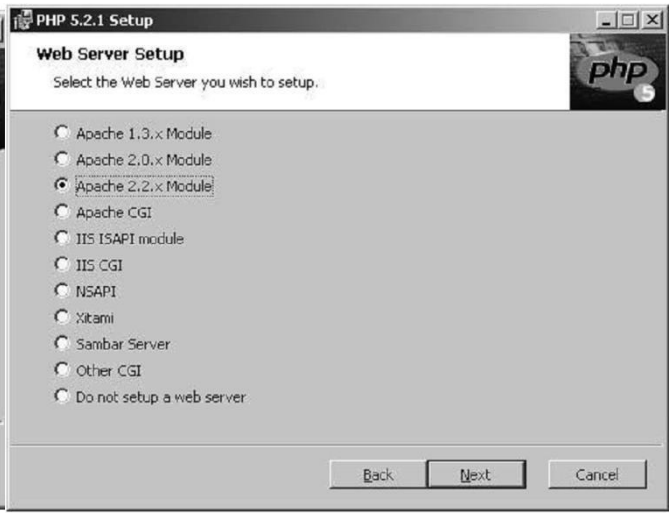
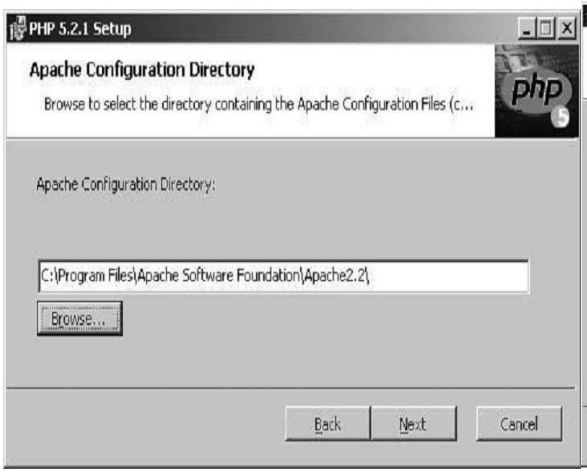


Figure 1.13 ***Figure 1.14***

6. The Web Server Setup dialog appears as shown in Figure 1.14. Select ―Apache 2.2.xModule" and click Next. If you were using a different web server like IIS, you could select that option here.
7. The Apache Configuration Directory dialog specifies where you installed Apache so that the installer can set up the Apache configuration to use PHP for you. It should be similar to **C:\Program Files\Apache Software Foundation\Apache2.2**, as shown in Figure 1.15.



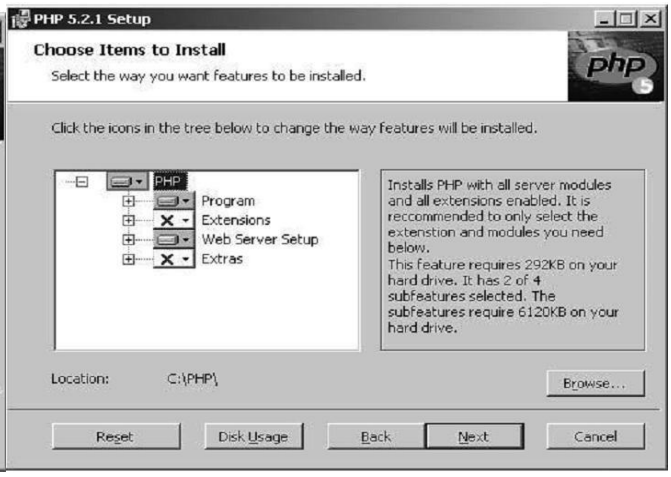


Figure 1.15 ***Figure 1.16***

8. Figure 1.16 shows the ―Choose Items to Install" dialog. The defaults on this dialog areall OK. If you changed the base install directory, you may also need to change it here.Click Next.
9. Click Install on the ―Ready to install" screen to confirm the installation.
10. Click Yes to confirm configuring Apache when the dialog shown in Figure 1-17 appears.

Figure 1.17: Dialog confirming that the installer will configure Apache 11. Click OK on the Apache Config dialog to acknowledge the successful Apache update for **―httpd.conf‖**.

12. Click OK on the Apache Config dialog to acknowledge the successful Apache update for **mime.types**.
13. The Successful Installation dialog appears.
14. Restart the Apache server by selecting Start → All Programs → Apache HTTP Server

2.x.x → Control Apache Server → Restart, so that it can read the new configuration directives that the PHP installer placed in the **httpd.conf** configuration file. This file tells Apache to load the PHP process as a module. Alternatively, in the system tray, double-click the Apache icon and click the Restart button.

1.4.2 Installing Apache for Linux/UNIX

To download the Apache distribution for Linux, start at the Apache Server Web site, **http://httpd.apache.org/**,and follow the link to Download. The current version is 2.2.4, and I prefer *.tar.gz files, so the file used as an example throughout this section is httpd-2.2.4.tar.gz.

1. Type **cp httpd-2.2.4.tar.gz /usr/local/** and press Enter to copy the Apache installation file to the **/usr/local/src** directory.
2. Go to **/usr/local/src** by typing **cd/usr/local/src** and pressing Enter.
3. Unzip the Apache installation le by typing **gunzip httpd-2.2.4.tar.gz** and pressing Enter.
4. Extract the files by typing **tar -xvf httpd-2.2.4.tar** and pressing Enter. A directory structure will be created, and you'll be back at the prompt. The parent directory will be**/usr/local/src/httpd2.0.49/**.
5. Enter the parent directory by typing **cd httpd-2.2.4** and pressing Enter.
6. Type the following and press Enter to prepare to build Apache:

./configure --prefix=/usr/local/apache2 --enable-module=so

The configuration script will run through its process of checking your configuration and creating make files, and then it will put you back at the prompt.

7. Type **make** and press Enter. This second step of the installation process will produce many lines of output on your screen. When it is finished, you will be back at the prompt.
8. Type **make** install and press Enter. This final step of the installation process will again produce many lines of output on your screen. When it is finished, you will be back at the prompt.

If your installation process produces any errors up to this point, go through the process again or check the Apache Web site for any system-specific notes. In the next section, you'll make some minor changes to the Apache conguration file before you start Apache for the first time.

Configuring Apache on Linux

To run a basic installation of Apache, the only changes you need to make are to the server name, which resides in the master configuration file called **httpd.conf**. This file lives in the**conf** directory, within the Apache installation directory. So if your installation directory is**/usr/local/apache2/**, the configuration files will be in **/usr/local/apache2/conf**/.

To modify the basic configuration, most importantly the server name, open the **httpd.conf**file with a text editor and look for a heading called Main server configuration. You will find two important sections of text.

We are going to change the values in the configuration file so that Apache knows where to find things and who to send complaints to. The **ServerAdmin**, which is you, is simply the e-mail address that people can send mail to in reference to your site. The **ServerName** is what Apache uses to route incoming requests properly.

1. Change the value of **ServerAdmin** to your e-mail address.

2. Change the value of **ServerName** to something accurate and remove the preceding # so that the entry looks like this:

   ServerName somehost.somedomain.com

3. Save the file.

Installing PHP for Linux

To download the PHP source distribution, visit the Downloads page at the PHP Web site:**www.php.net/downloads.php**.

1. The current source code version is 6.0.0, and that version number will be used in thefollowing steps.
2. Once downloaded to your system, type **cp php-6.0-dev.tar.gz /usr/local/src/** and press Enter to copy the PHP source distribution to the **/usr/local/src/** directory.
3. Go to **/usr/local/src/** by typing **cd /usr/local/src/** and pressing Enter.
4. Unzip the source file by typing **gunzip php-6.0-dev.tar.gz** and pressing Enter.
5. Extract the files by typing **tar -xvf php-6.0-dev.tar** and pressing Enter. This will createa directory structure and then put you back at the prompt. The parent directory will be**/usr/local/src/php-6.0.0/**.
6. Enter the parent directory by typing **cd php-6.0-dev** and pressing Enter.
7. Type the following and press Enter to prepare to build PHP:

   ./configure --prefix=/usr/local/php5 --with-mysql=/usr/local/mysql/
   --with-apxs2=/usr/local/apache2/bin/apxs

The configuration script will run through its process of checking your configuration and creating makefiles and then will put you back at the prompt.

8. Type **make** and press Enter. This second step of the installation process will produce many lines of output on your screen. When it is finished, you will be back at the prompt.
9. Type **make** install and press Enter. This final step of the installation process will produce many lines of output on your screen. When it is finished, you will be back at the prompt.

Now, to get a basic version of PHP working with Apache, all you need to do is to make a few modifications to the **httpd.conf**file.

Configuring Apache to Use PHP

The installation process will have placed a module in the proper place within the Apache directory structure. Now you must make some modifications to the **httpd.conf**file before starting up Apache with PHP enabled.

1. Open the **httpd.conf**file in your text editor of choice.
2. Look for the following line, which will have been inserted into the file by the installation process:
   LoadModule php6_module modules/libphp6.so

You want this line to be uncommented, so ensure that it is (as shown).

3. Look for the following lines:

   # AddType allows you to add to or override the MIME configuration #
   filemime.types for specific file types.
   #AddType application/x-tar .tgz

4. Add to these lines the following:

   AddType application/x-httpd-php .phtml .php

5. Save and close the **httpd.conf**file.

1.4.3 Installing IIS and PHP on Windows

Installing IIS Services:

a) Install windows in to your machine.
b) After installing windows, go to control panel.
c) In control panel, double click add/remove programs.
d) In the left part of the window, you can see the option add/remove windows component .click on add/remove window components.
e) Now, you can see the internet information services as a checkbox, initially unchecked.
f) Select that checkbox and click ok, automatically the system will ask for windows CD (windows XP only ie., Win7 won‘t ask for any CD),insert windows CD & click ok. It will install IIS on your system.
g) After installation, open control panel, double click on administration tools, double click on internet services manager icon. This opens the internet information services window (or)type "inetmgr" at start & run command.
h) Place the documents that will be requested from IIS either in the default directory i.e., c:\onetpub\wwwroot)in a virtual directory.
i) A virtual directory is an alias for an existing directory that resides on the local machine or on the network.
j) In the IIS window, the left pane contains the web server directory structure.
k) The name of the machine running IIS is listed under IIS.
l) Clicking the # symbol to the left of the machine name displays default FTP site, default web site, default SMTP virtual server.
m) FTP site is a file protocol site. the default website is an HTTP site.
n) The default SMTP virtual server allows you to create a simple mail transfer protocol(SMTP)sever for sending electronic mail(e-mail).
o) Expand the default website directory by clicking '+'.The default web server sub directories are virtual directories.
p) In this directory, we will create a virtual directory for the HTTP web site. to create a virtual directory with in this directory, right click on default web site, select "new", then virtual directory. this starts the virtual directory creation wizard.
q) In the virtual directory alias dialog, enter the name for the virtual directory & click next.
r) In the website content directory dialog, enter the path for the directory containing the documents that client will view. click next.
s) The access permissions dialog presents the virtual directory security level choices, includes read, run script, execute ,write, browse.

Installing & configuring PHP in windows with IIS Server:-

1. Download php zip file from php.net
2. Install IIS in the system.
3. Extract all of the files from the downloaded zip file into c:\php.
4. Php5 includes a CGI Executable as well as he server module. the DLL's needed for their executable scan be found in the root of the php folder(c:\php).
5. phpts.dll needs to be available to the web browser to do this , you have 3 options.

a) copy php5ts.dll to the web servers directly(c:/input/user).
b) copy php5ts.dll to the windows system directory(sys32).
c) add to path directory path environment variables.

6. Right click on my computer and choose properties.
7. Select advanced tab click the environment variables button.
8. In the system variables click on "path" select edit
9. Go to the end of the line add a semicolon(;) if there is not one already add c:\php and click ok.
10. Restart the computer once to effect the changes.
11. Now we want to setup config file for php, php.ini.
12. In c:\php you will find two files named php.ini_production, php.ini_development. selectphp.ini_production.
13. Rename it to php.ini.
14. Open php.ini search doc_root = c:\inetpub\wwwroot.
15. cgi.force_redirect = 0 and save it.

Move to php with IIS:-

1. Open IIS.
2. Expand the IIS directories, right click on default websites under home directory tab , set execute permissions to scripts only.
3. Click on configuration click and button.
4. Browse the path to php i.e..to insert php5 is api.dll. i.e..(c:\php\php5 is api.dll) set extension to phpok.
5. Under isapi filters, add, set filter name to php. Set executable to c:\php\php5 isapi.dll okapply.
6. Open note pad type: <?php echo phpinto(); ?>. Save it as phpinfo.php in c:\inetpub\wwwroot. Open web browser as type http://localhost/phpinfo.php.

1.4.4 Installing a XAMPP on Linux

If you know much about Linux, you may have already set up and installed PHP and MySQL. Ifnot, your best bet is probably to look at XAMPP for Linux, which is available athttp://apachefriends.org/en/xampp-linux.html.

The process is relatively simple. After downloading, go to a Linux shell and log in as thesystem administrator (root) by typing:**su**

Enter your system administration password. Many desktop Linux systems allow you to useyour personal account's password for the administration password. Some systems, includingthe popular Ubuntu, encourage you not to use su to log in as root, but to precede each systemadministration command with **sudo** instead. You'll know what to do if you've performed anyadministrative tasks on your system. Now extract the downloaded archive file to /opt with thefollowing command (inserting the appropriate filename if the version you downloaded is a laterversion):

tarxvfz xampp-linux-1.6.8a.tar.gz -C /opt

Any XAMPP version that was already installed will be overwritten by this command. Oncethe command finishes, XAMPP will be installed below the /opt/lampp directory. To start it,enter the following: **/opt/lampp/lampp start**

You should now see something like this on your screen:

Starting XAMPP 1.6.8a...
LAMPP: Starting Apache...
LAMPP: Starting MySQL...
LAMPP started.
Ready. Apache and MySQL are running.
Now you are ready to test the setup. Type the following URL into your web browsers addressbar:http://localhost

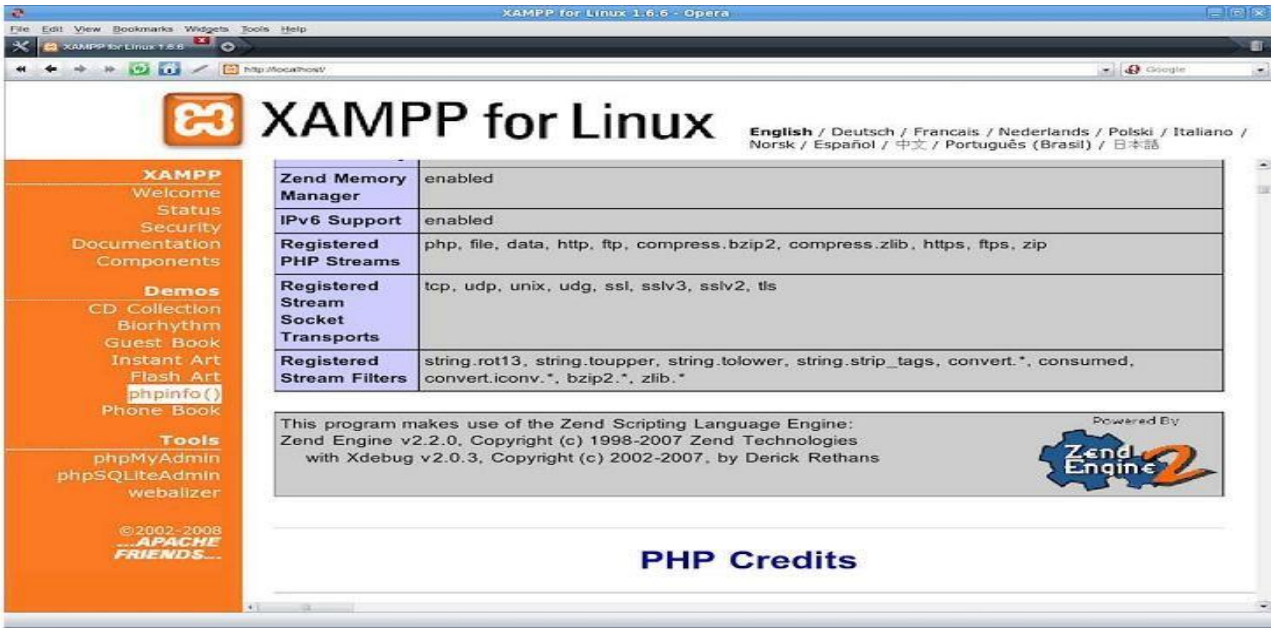


Figure 1.18: XAMPP for Linux, installed and running

1.4.5 Installing XAMPP on Windows

The following steps cover installing XAMPP on Windows:

1. Download the Basic Package XAMPP MSI installer found at http://www.apachefriends.org/en/xampp-windows.html
2. Double-click the MSI installer file on your desktop, and you‘ll see the installer shown inFigure 1.19
3. Select English and click the OK button.
4. The Setup Wizard appears as shown in Figure 1.20. Click Next.
5. The dialog shown in Figure 1.21 is displayed. Click Next to accept the default installationdirectory.

Figure 1.19

Figure 1.20

Figure 1.20: The Xampp Setup Wizard

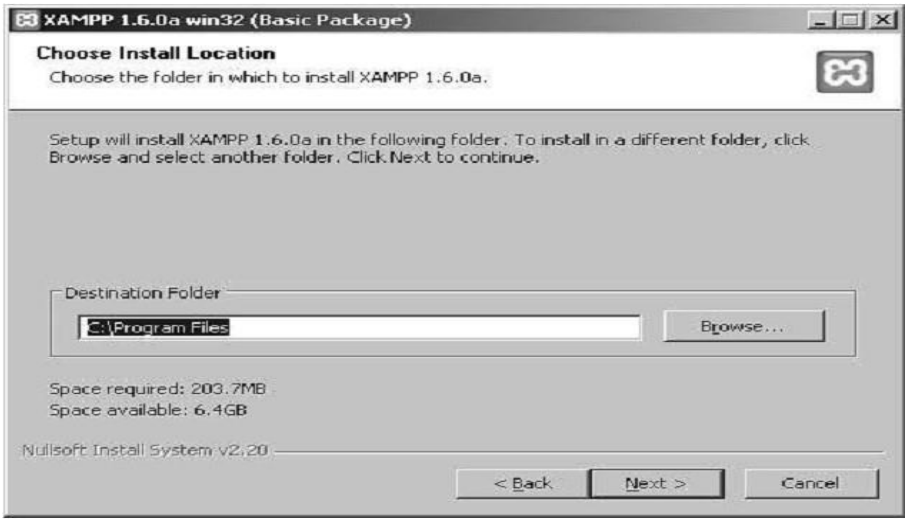


Figure 1.21: Select the installation directory

6. The XAMPP Options dialog displays, as shown in Figure 1.22. Leave the Service Section checkboxes unchecked so you don‘t install the components as services; instead, you'll start them from the Control Panel. Click Install.

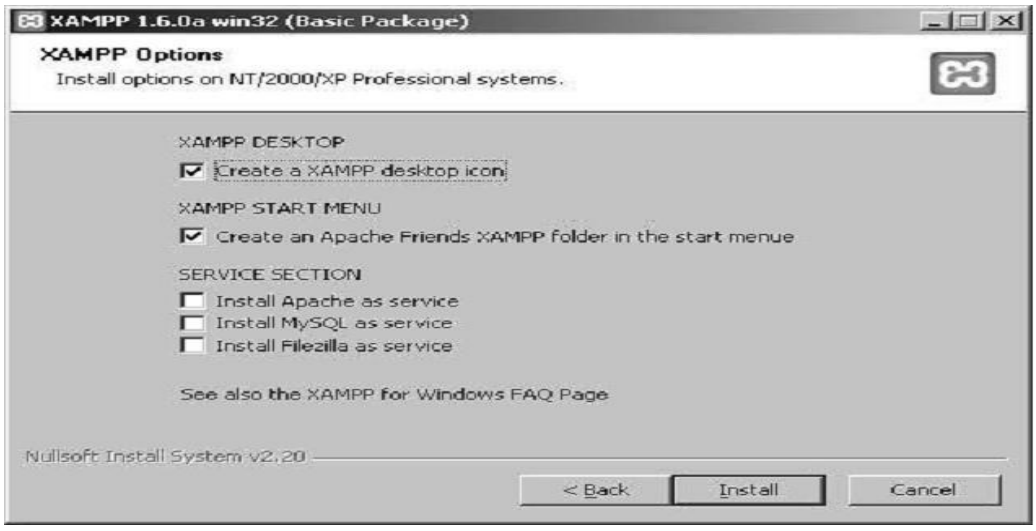


Figure 1.22: Choose your installation options 7.

The Completing the XAMPP Setup Wizard displays. Click Finish.

8. The option to start the Control Panel displays, Click Yes.
9. The Control Panel launches, as shown in Figure 1.23. The Control Panel can start and stop the services, as well as aid in their configuration.

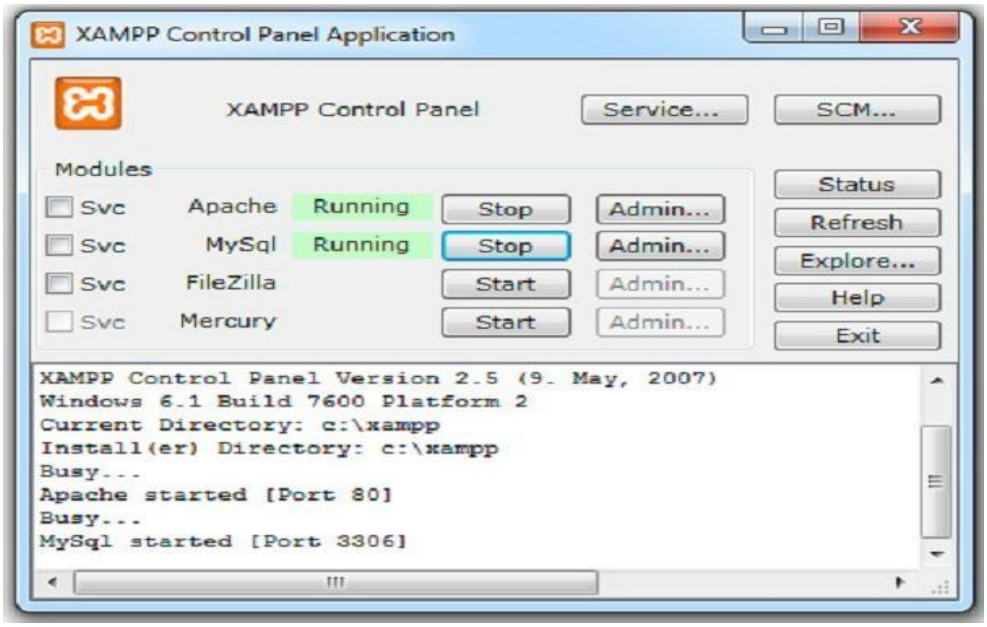


Figure 1.23: The Control Panel starts and stops the components

1.4.6 Installing WAMP

If you are installing **WampServer 2.1d**, then these following steps will help you that how to install the WampServer 2.1d in your computer with windows 7. This server can be found fordownload at official web page WampServer.

1. It is the time to install WampServer on our windows. You will receive a Security Warningafter opening WampServer file. It is absolutely normal to run WampServer setup on windows.(Fig 1.24)

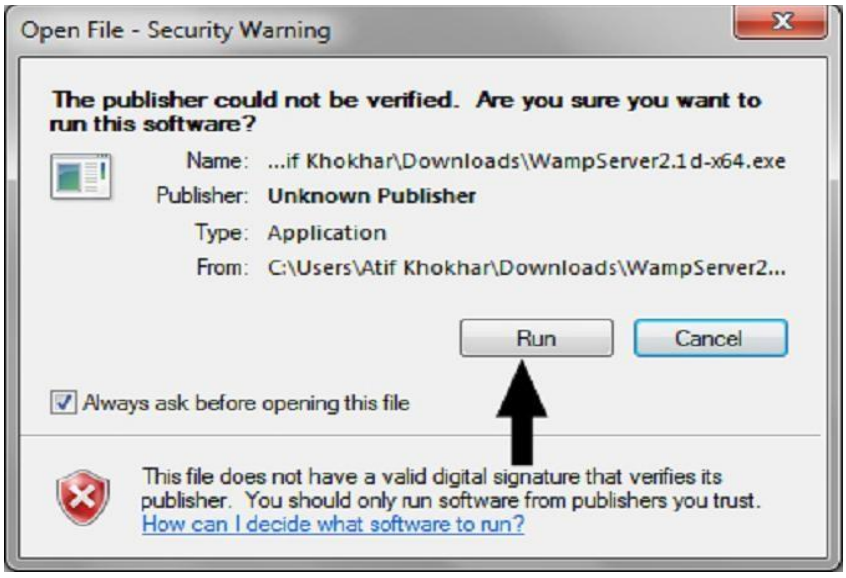
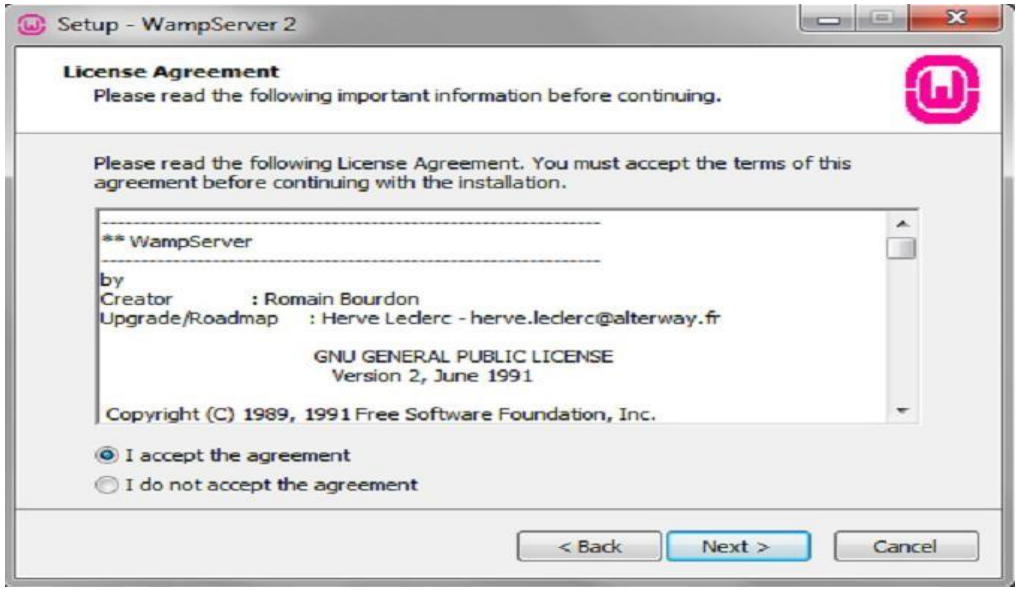


Figure 1.24: Instalation Starting of WampServer

2. You will see a standard setup wizard of windows after clicking Run button on securitywarningdialog(Fig 1.25)

Figure 1.25: WampServer 2 Setup Wizard

3. You have to agree the license of WampServer before selecting installation destination atyour windows machine.(Fig. 1.26)



4. It is very important step of WampServer installation. I will recommend installingWampServerat the drive other than Windows 7 installation. Suppose your Windows 7 is install in Cdrive so you should install WampServer on D, E or any other location in hard drive exceptC drive.

I am going to install WampServer in D drive. Now you can click on Next button afterselecting installation location for WampServer 2.1d.(Fig. 1.27)

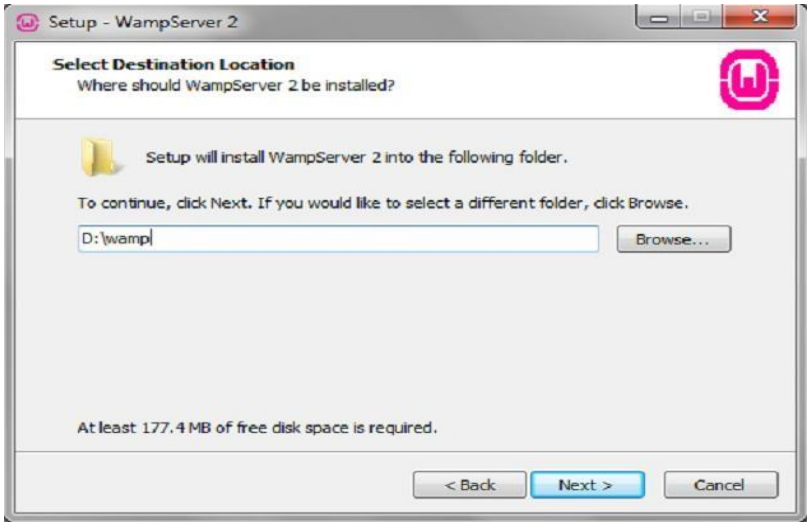


Figure 1.27: Select Destination Location of WampServer

5. When you click on the Next button then a Select Additional Tasks dialog will appear onyour screen, if you would like setup to perform while installing WampServer 2. You cancheck following options,

- Create a Quick Launch icon
- Create a Desktop icon

I have not interested to create any icon in the above locations, but you can do. You willbe at ―Ready to Install" window after clicking Next button.(Figure 1.28)

6. Setup is now ready to begin installing WampServer 2.1d on your computer. Click on Install button to start installation of WampServer 2.1d.(Figure 1.29)

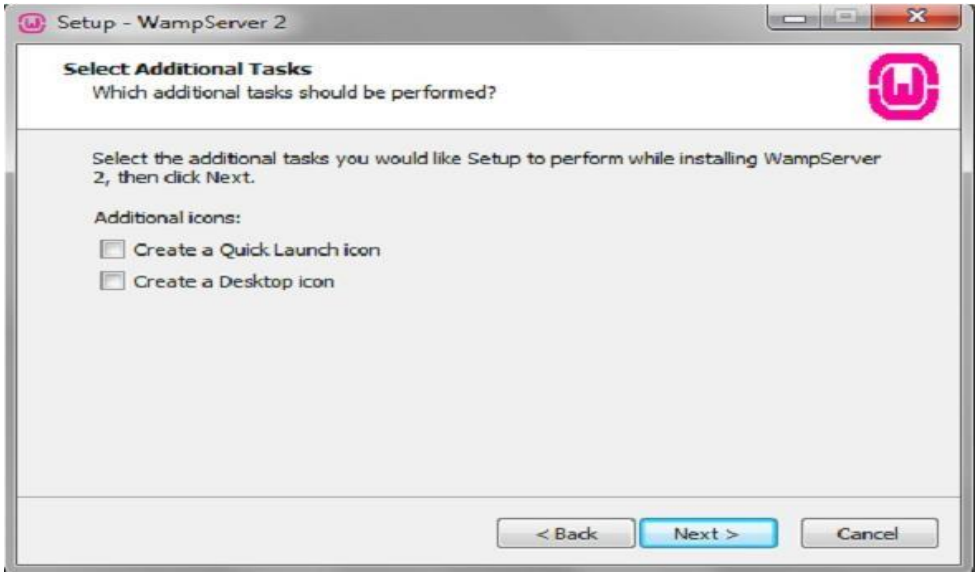


Figure 1.28: Select Additional Tasks

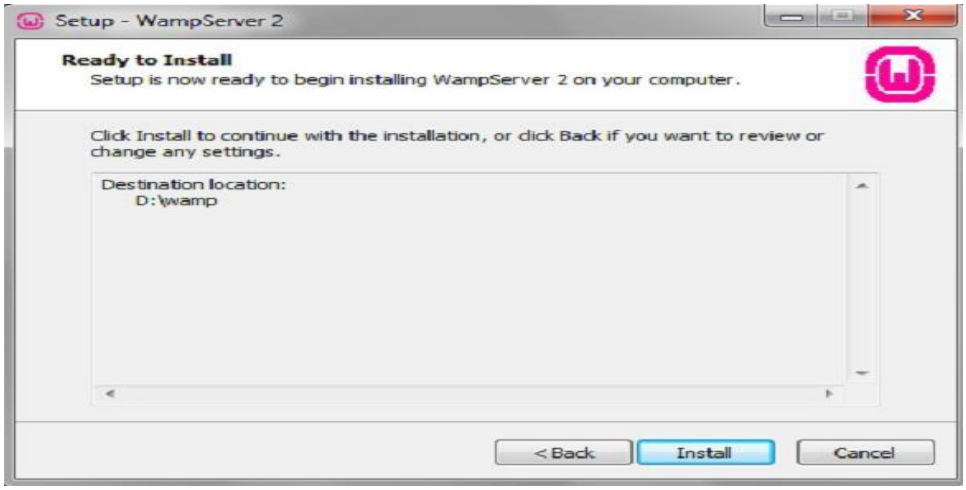


Figure 1.29: WampServer 2.1d Ready to Install

7. Now your WampServer is starting to install in your computer.(Figure 1.30)

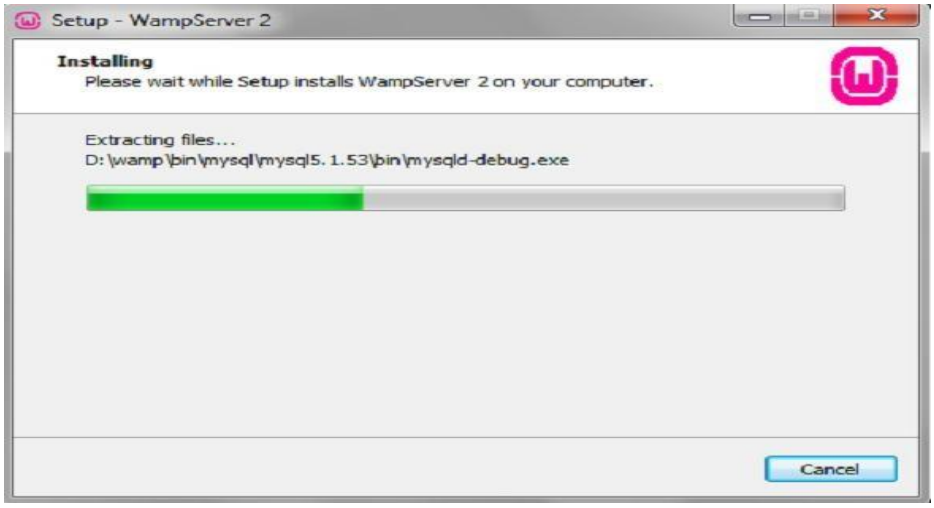


Figure 1.30: WampServer Installing

8. You will receive a dialog for choosing your default browser for WampServer. You canchoose your favorite browser for WampServer as default, or simply click —Open" if you are not sure about the installation or executable files of your favorite browser.(Figure 1.31)

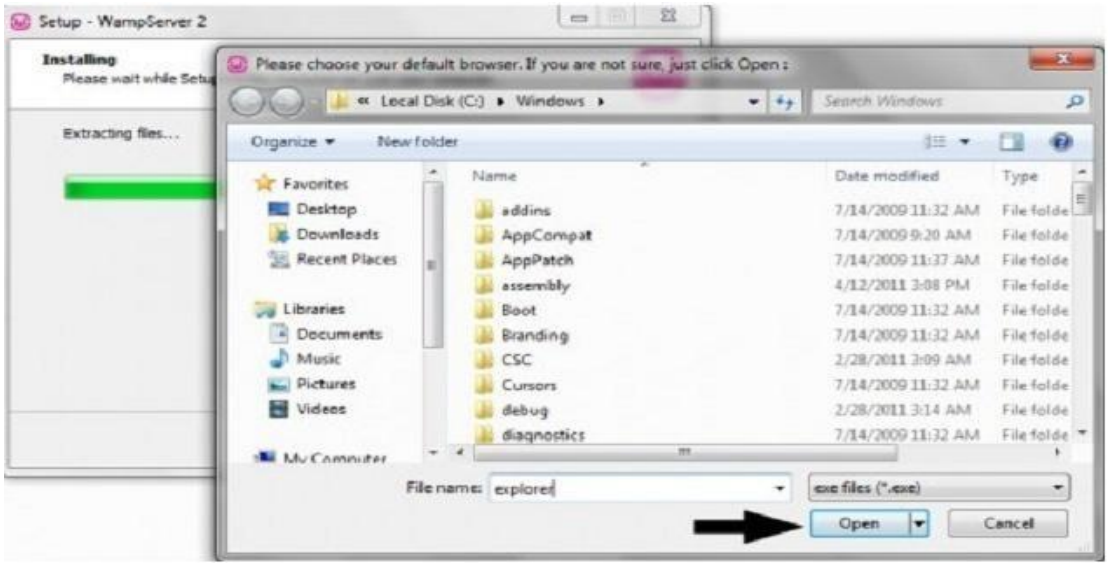


Figure 1.31: Choice of Default Browser

9. WampServer installation has completed now and setup will guide you for Apache configurations in the next steps.(Figure 1.32)

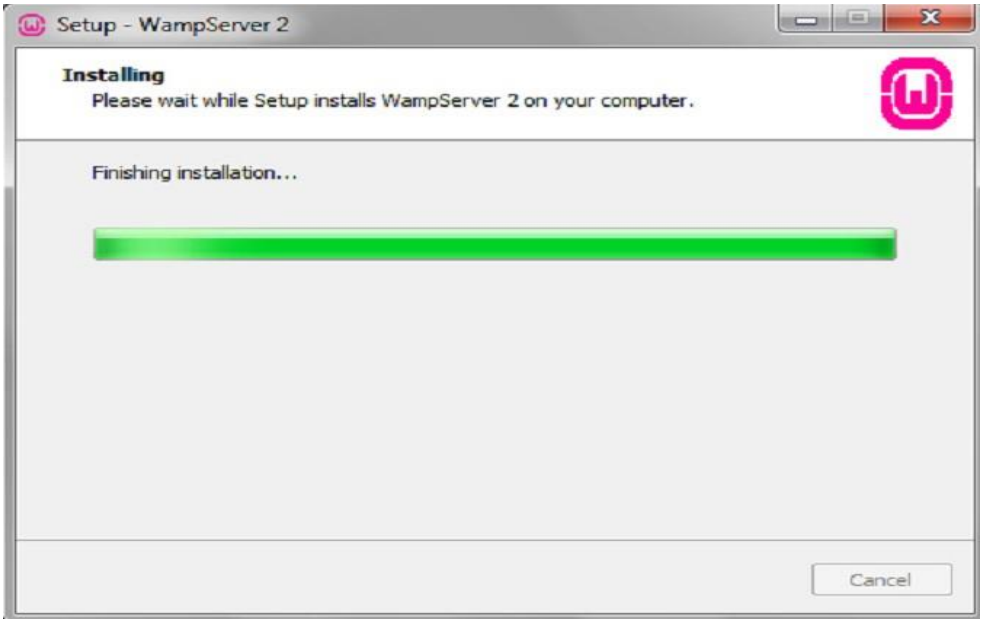


Figure 1.32: Complete the Installation

10. You will notice a ―Windows Firewall" standard dialog while configuring Apache byWampServer.(You may not observe this, if your windows firewall is not active). Click on ―Allow Access"by leaving default options as such to proceed for PHP mail parameters.(Figure 1.33)

Figure 1.33: Apache HTTP Server

11. After allowing access to Apache server, you are at SMTP server configuration dialog. Youcan specify the SMTP server and the address mail to be used by PHP when using thefunction mail(). I will recommend the following values,

- SMTP: localhost
- Email: Your email address.

Click Next after putting the above values for the installation. fiinal dialog.(Figure 1.34)

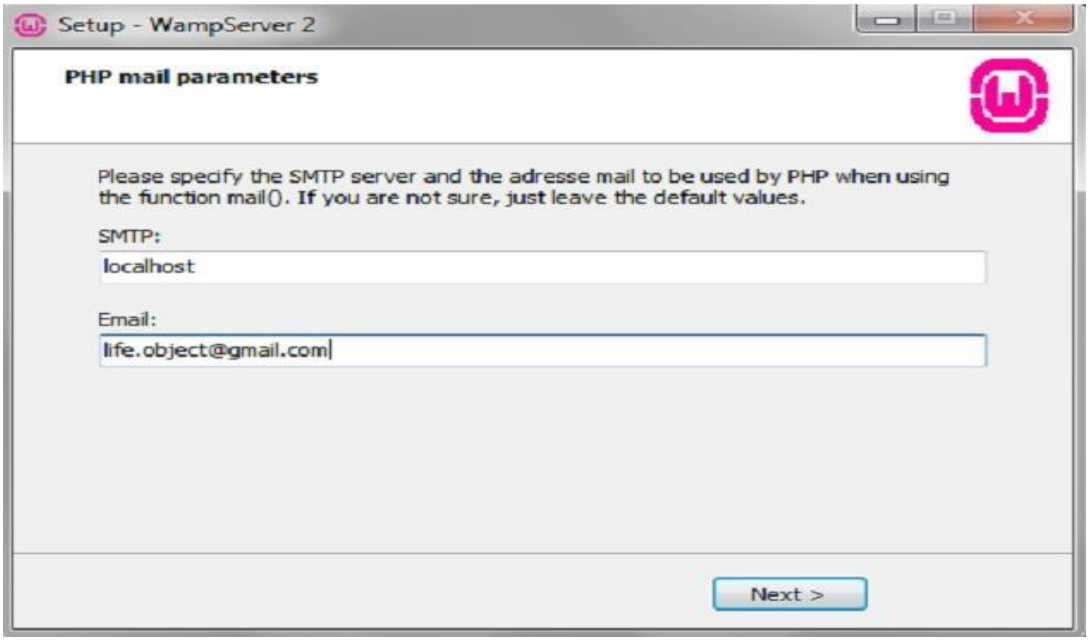


Figure 1.34: PHP Mail Parameters

Figure 1.35: WampServer 2 Setup Wizard Completion

12. You have successfully installed WampServer 2.1 d along with Apache, MySql, PHP, phpMyAdmin and SqlBuddy at your computer.

Click ―Finish" tostartWampServer along with other services. Leave ―Launch WampServer 2 now" check-box checked to start WampServer automatically after installation.(Fig1.35)

1.5 HTML Introduction

HTML is the standard markup language for creating Web pages.

HTML stands for Hyper Text Markup Language

- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on A Simple HTML Document

Example

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

Example Explained

- The <!DOCTYPE html> declaration defines this document to be HTML5
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the document
- The <title> element specifies a title for the document
- The <body> element contains the visible page content
- The <h1> element defines a large heading
- The element defines a paragraph

HTML Tags

HTML tags are element names surrounded by angle brackets:

<tagname>content goes here...</tagname>

- HTML tags normally come **in pairs** like and
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

HTML Versions

Since the early days of the web, there have been many versions of HTML:

Version	**Year**
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

HTML Documents

All HTML documents must start with a document type declaration: **<!DOCTYPE html>**.

The HTML document itself begins with **<html>** and ends with **</html>**.
The visible part of the HTML document is between **<body>** and **</body>**.

Example

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

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

</body>
</html>
```

HTML Headings

HTML headings are defined with the **<h1>** to **<h6>** tags.
<h1> defines the most important heading. <h6> defines the least important heading: **Example**

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

HTML Paragraphs

HTML paragraphs are defined with the **<p>** tag:

Example

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

HTML Links

HTML links are defined with the **<a>** tag:

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

<a href="https://www.w3schools.com">This is a link</a>

</body>
</html>
```

The link's destination is specified in the **href attribute**.
Attributes are used to provide additional information about HTML elements.

HTML Images

HTML images are defined with the **<img>** tag.
The source file (src), alternative text (alt), width, and height are provided as attributes:

Example

```
<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
```

HTML Elements

An HTML element usually consists of a **start** tag and **end** tag, with the content inserted in between:

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

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

My first paragraph.

Empty HTML Elements

HTML elements with no content are called empty elements.

 is an empty element without a closing tag (the
 tag defines a line break).

Empty elements can be "closed" in the opening tag like this:
.

HTML5 does not require empty elements to be closed. But if you want stricter validation, or if you need to make your document readable by XML parsers, you must close all HTML elements properly.

HTML Attributes

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

The width and height Attributes

Images in HTML have a set of size attributes, which specifies the width and height of the image:

Example

<img src="img_girl.jpg" width="500" height="600">

The style Attribute

The style attribute is used to specify the styling of an element, like color, font, size etc.

Example

I am a paragraph

HTML Headings

Headings are defined with the <h1> to <h6> tags.

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

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

Headings

Each HTML heading has a default size. However, you can specify the size for any heading with the style attribute:

Example

<h1 style="font-size:60px;">Heading 1</h1>

HTML Horizontal Rules

The <hr> tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule.

The <hr> element is used to separate content (or define a change) in an HTML page: Example

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

View HTML Source Code:

To find out, right-click in the page and select "View Page Source" (in Chrome) or "View Source" (in IE), or similar in other browsers. This will open a window containing the HTML source code of the page.

Inspect an HTML Element:

Right-click on an element (or a blank area), and choose "Inspect" or "Inspect Element" to see what elements are made up of (you will see both the HTML and the CSS). You can also edit the HTML or CSS on-the-fly in the Elements or Styles panel that opens.

HTML Line Breaks

The HTML **
** element defines a **line break**.

Use
 if you want a line break (a new line) without starting a new paragraph:

Example

```
<p>This is<br>a paragraph<br>with line breaks.</p>
```

The HTML <pre> Element

The HTML <pre> element defines preformatted text.

The text inside a <pre> element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks:

Example

```
<pre>
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.

  Oh, bring back my Bonnie to me.
</pre>
```

HTML Formatting Elements

In the previous chapter, you learned about the HTML **style attribute**.

HTML also defines special **elements** for defining text with a special **meaning**.

HTML uses elements like <b> and <i> for formatting output, like **bold** or *italic* text.

Formatting elements were designed to display special types of text:

HTML Text Formatting Elements

Tag	Description
<b>	**Defines bold text**
****	**Defines emphasized text**
<i>	**Defines italic text**
<small>	**Defines smaller text**

	Defines important text
<sub>	**Defines subscripted text**
<sup>	**Defines superscripted text**
<ins>	**Defines inserted text**
<del>	**Defines deleted text**
<mark>	**Defines marked/highlighted text**

HTML <b> Elements

The HTML **<b>** element defines **bold** text, without any extra importance.

Example

```
<!DOCTYPE html>
<html>
<body>
<p>This text is normal.</p>
<p><b>This text is bold.</b></p>
</body>
</html>
```

HTML Comment Tags

You can add comments to your HTML source by using the following syntax:

<!-- Write your comments here -->

Notice that there is an exclamation point (!) in the opening tag, but not in the closing tag.

Note: Comments are not displayed by the browser, but they can help document your HTML source code.

With comments you can place notifications and reminders in your HTML:

Example

```
<!-- This is a comment -->

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

<!-- Remember to add more information here -->
```

HTML colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

Color Names

In HTML, a color can be specified by using a color name:

- Tomato
- Orange
- DodgerBlue
- MediumSeaGreen
- Gray

- SlateBlue
- Violet
- LightGray

Example

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

Text Color

Example

<h1 style="color:Tomato;">Hello World</h1>
Lorem ipsum...
Ut wisi enim...

Border Color

Example

<h1 style="border:2px solid Tomato;">Hello World</h1>
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
<h1 style="border:2px solid Violet;">Hello World</h1>

Styling HTML with CSS

CSS stands for **C**ascading **S**tyle **S**heets.

CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**.

CSS **saves a lot of work**. It can control the layout of multiple web pages all at once.

CSS can be added to HTML elements in 3 ways:

- **Inline** - by using the style attribute in HTML elements
- **Internal** - by using a <style> element in the <head> section
- **External** - by using an external CSS file

The most common way to add CSS, is to keep the styles in separate CSS files. However, here we will use inline and internal styling, because this is easier to demonstrate, and easier for you to try it yourself.

Tip: You can learn much more about CSS in our CSS Tutorial.

Inline CSS

An inline CSS is used to apply a unique style to a single HTML element.

An inline CSS uses the style attribute of an HTML element.

This example sets the text color of the <h1> element to blue:

Example

<h1 style="color:blue;">This is a Blue Heading</h1>

Internal CSS

An internal CSS is used to define a style for a single HTML page.

An internal CSS is defined in the <head> section of an HTML page, within a <style> element:

Example

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

```
h1   {color: blue;} p
{color: red;}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

External CSS

An external style sheet is used to define the style for many HTML pages.

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

To use an external style sheet, add a link to it in the <head> section of the HTML page:

Example

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

An external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a .css extension.

Here is how the "styles.css" looks:

```
body
{
   background-color: powderblue;
} h1
{
   color: blue;
} p
{
   color: red; }
```

CSS Fonts

The CSS **color** property defines the text color to be used.

The CSS **font-family** property defines the font to be used.

The CSS **font-size** property defines the text size to be used.

Example

```
<!DOCTYPE html>
<html>
<head> <style> h1 {
color: blue;     font-
family: verdana;
font-size: 300%;
} p   {        color:  red;
font-family:     courier;
font-size: 160%;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

CSS Border

The CSS **border** property defines a border around an HTML element:

Example

```
p {
   border: 1px solid powderblue;
}
```

CSS Margin

The CSS **margin** property defines a margin (space) outside the border: Example

```
p {
   border:     1px     solid     powderblue;
margin: 50px;
}
```

The id Attribute

To define a specific style for one special element, add an id attribute to the element:

```
<!DOCTYPE html>
<html>
<head>
<style> #p01
{
   color: blue;
}
</style>
</head>
<body>
```

This is a paragraph.
This is a paragraph.
I am different.
</body>
</html>

The class Attribute

To define a style for a special type of elements, add a class attribute to the element:
I am different then define a style for
the elements with the specific class:

Example

```
p.error {
                                                    color: red;
}
```

External References

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

Example

<link rel="stylesheet" href="https://www.w3schools.com/html/styles.css">
This example links to a style sheet located in the html folder on the current web site: **Example**
<link rel="stylesheet" href="/html/styles.css">
This example links to a style sheet located in the same folder as the current page:

Example

<link rel="stylesheet" href="styles.css">
HTML Links - Hyperlinks
HTML links are hyperlinks.
You can click on a link and jump to another document.
When you move the mouse over a link, the mouse arrow will turn into a little hand.
Note: A link does not have to be text. It can be an image or any other HTML element. **HTML Links - Syntax**
In HTML, links are defined with the **<a>** tag:
link text
Example
Visit our HTML tutorial
The **href** attribute specifies the destination address (https://www.w3schools.com/html/) of the link.
The **link text** is the visible part (Visit our HTML tutorial).
Clicking on the link text will send you to the specified address.
Note: Without a forward slash on subfolder addresses, you might generate two requests to the server.
Many servers will automatically add a forward slash to the address, and then create a new request.

HTML Images

Images can improve the design and the appearance of a web page.
Example
<img src="pulpitrock.jpg" alt="Mountain View">
Image Maps

Use the <map> tag to define an image-map. An image-map is an image with clickable areas. In the image below, click on the computer, the phone, or the cup of coffee:

```
<!DOCTYPE html>
<html>
<body>
<p>Click on the computer, the phone, or the cup of coffee to go to a new page and read more about the topic:</p>
<img src="workplace.jpg" alt="Workplace" usemap="#workmap" width="400" height="379">
<map name="workmap">
  <area shape="rect" coords="34,44,270,350" alt="Computer" href="computer.htm">
  <area shape="rect" coords="290,172,333,250" alt="Phone" href="phone.htm">
  <area shape="circle" coords="337,300,44" alt="Cup of coffee" href="coffee.htm"> </map>
</body>
</html>
```

HTML Tables

Defining an HTML Table

An HTML table is defined with the **<table>** tag.

Each table row is defined with the **<tr>** tag. A table header is defined with the **<th>** tag. By default, table headings are bold and centered. A table data/cell is defined with the **<td>** tag. **Example**

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

<table style="width:100%">
  <tr>
    <th>Firstname</th>
```

```
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>80</td>
  </tr>
</table>

</body>
</html>
```

Note: The <td> elements are the data containers of the table. They can contain all sorts of HTML elements; text, images, lists, other tables, etc.

HTML Table - Adding a Border

If you do not specify a border for the table, it will be displayed without borders.

A border is set using the CSS **border** property:

```
<!DOCTYPE html>
<html>
<head>
<style> table,
th, td {
   border: 1px solid black;
}
</style>
</head>
<body>

<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
```

```
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>80</td>
  </tr>
</table>

</body>
</html>
```

Output:

Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94
John	Doe	80

HTML Table - Collapsed Borders

If you want the borders to collapse into one border, add the CSS **border-collapse** property:

Example

```
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
```

HTML Table - Adding Cell Padding

Cell padding specifies the space between the cell content and its borders.

If you do not specify a padding, the table cells will be displayed without padding.

To set the padding, use the CSS **padding** property: Example

```
th, td {
padding: 15px;
}
```

HTML Table - Left-align Headings

By default, table headings are bold and centered.

To left-align the table headings, use the CSS **text-align** property: **Example**

```
th {
```

```
  text-align: left;
}
```

Adding a Caption

To add a caption to a table, use the **<caption>** tag:

Example

```
<table style="width:100%">
  <caption>Monthly savings</caption>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>February</td>
    <td>$50</td>
  </tr>
</table>
```

A Special Style for One Table

To define a special style for a special table, add an **id** attribute to the table:

Example

```
<table id="t01">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Now you can define a special style for this table:

```
table#t01 {     width: 100%;
  background-color: #f1f1c1;
}
```

HTML Lists HTML List Example An Unordered List:

- Item
- Item

- Item
- Item

An Ordered List:

1. First item
2. Second item
3. Third item
4. Fourth item

Unordered HTML List

An unordered list starts with the **<ul>** tag. Each list item starts with the **<li>** tag. The list items will be marked with bullets (small black circles) by default:

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li> </ul> nordered HTML List -
```

Choose List Item Marker

The CSS **list-style-type** property is used to define the style of the list item marker:

Value	Description
disc	Sets the list item marker to a bullet (default)
circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

Example - Disc

```
<ul style="list-style-type:disc">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Ordered HTML List

An ordered list starts with the **<ol>** tag. Each list item starts with the **<li>** tag. The list items will be marked with numbers by default:

Example

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Ordered HTML List - The Type Attribute

The **type** attribute of the <ol> tag, defines the type of the list item marker:

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

Numbers:

```
<ol type="1">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

HTML Description Lists

HTML also supports description lists.

A description list is a list of terms, with a description of each term.

The **<dl>** tag defines the description list, the **<dt>** tag defines the term (name), and the **<dd>** tag describes each term:

Example

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

Block-level Elements

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

The element is a block-level element.

Example

```
<div>Hello</div>
<div>World</div>
```

Block level elements in HTML:

```
<address>
<article>
<aside>
<blockquote>
<canvas>
<dd>
<div>
<dl>
```

<dt>
<fieldset>
<figcaption>
<figure>
<footer>
<form>
<h1>-<h6>
<header>
<hr>

<main>
<nav>
<noscript>

<output>
<p>
<pre>
<section>
<table>
<tfoot> <video>

Inline Elements

An inline element does not start on a new line and only takes up as much width as necessary. This is an inline element inside a paragraph.

Example

Hello
World

Inline elements in HTML:

<a>
<abbr>
<acronym>
<b>
<bdo>
<big>

<button>
<cite>
<code>
<dfn>

<i>
<img>
<input>
<kbd>
<label>
<map>

<object>
<q>
<samp>
<script>
<select>
<small>

<sub>
<sup>
<textarea>
<time>
<tt>
<var>

The Element

The element is often used as a container for other HTML elements.

The element has no required attributes, but both **style** and **class** are common.

When used together with CSS, the element can be used to style blocks of content:

Example

```
<div style="background-color:black;color:white;padding:20px;">
  <h2>London</h2>
  <p>London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>
</div>
```

The Element

The element is often used as a container for some text.

The element has no required attributes, but both **style** and **class** are common.

When used together with CSS, the element can be used to style parts of the text:

Example

```
<h1>My <span style="color:red">Important</span> Heading</h1>
```

HTML The class Attribute

Using The class Attribute

The class attribute specifies one or more class names for an HTML element.

The class name can be used by CSS and JavaScript to perform certain tasks for elements with the specified class name.

Example

Using CSS to style all elements with the class name "city":

```
<style> .city
{
   background-color: tomato;
```

```
   color: white;
padding: 10px;
}
</style>

<h2 class="city">London</h2>
<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>
<p>Paris is the capital of France.</p>

<h2 class="city">Tokyo</h2>
<p>Tokyo is the capital of Japan.</p>
```

HTML Iframes

An iframe is used to display a web page within a web page.

Iframe - Set Height and Width

Use the **height** and **width** attributes to specify the size of the iframe.

The attribute values are specified in pixels by default, but they can also be in percent (like "80%").

Syntax

An HTML iframe is defined with the **<iframe>** tag:

<iframe src="*URL*"></iframe>

The **src** attribute specifies the URL (web address) of the inline frame page.

```
<!DOCTYPE html>
<html>
<body>
<iframe src="demo_iframe.htm" height="200" width="300"></iframe>
</body>
</html>
```

Iframe - Remove the Border

By default, an iframe has a border around it.

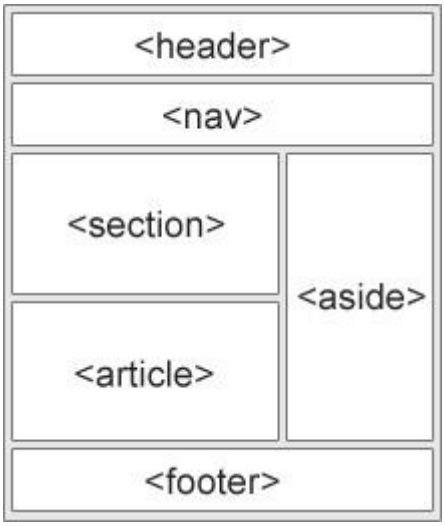
To remove the border, add the **style** attribute and use the CSS **border** property: Example

<iframe src="demo_iframe.htm" style="border:none;"></iframe>

HTML Layouts

Websites often display content in multiple columns (like a magazine or newspaper).

HTML5 offers new semantic elements that define the different parts of a web page:

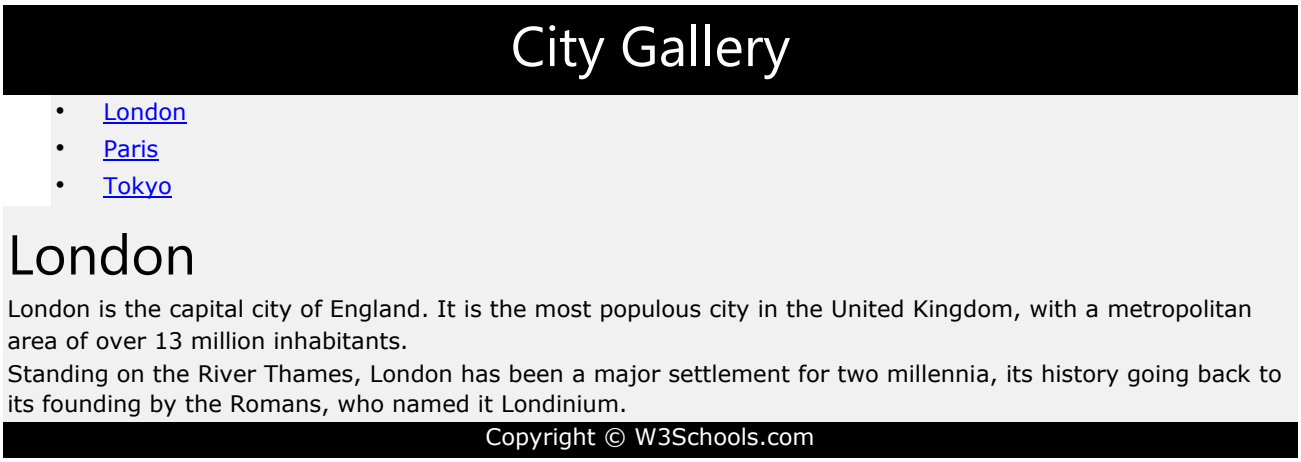


- <header> - Defines a header for a document or a section
- <nav> - Defines a container for navigation links
- <section> - Defines a section in a document
- <article> - Defines an independent self-contained article
- <aside> - Defines content aside from the content (like a sidebar)
- <footer> - Defines a footer for a document or a section
- <details> - Defines additional details
- <summary> - Defines a heading for the <details> element

CSS Floats

It is common to do entire web layouts using the CSS float property. Float is easy to learn - you just need to remember how the float and clear properties work. Disadvantages: Floating elements are tied to the document flow, which may harm the flexibility.

Float Example



Example:

```
<!DOCTYPE html>
<html>
<head> <style>
div.container {
width: 100%;
   border: 1px solid gray;
}

header, footer {     padding:
1em;     color: white;
background-color: black;
clear: left;     text-align:
center;
}
```

```
nav {
   float: left;
   max-width: 160px;
   margin:              0;
padding: 1em;
}

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

nav ul a {
   text-decoration: none;
}

article {
   margin-left: 170px;
border-left: 1px solid gray;
padding: 1em;     overflow:
hidden;
}
</style>
</head>
<body>

<div class="container">

<header>
  <h1>City Gallery</h1>
</header>

<nav>
 <ul>
   <li><a href="#">London</a></li>
   <li><a href="#">Paris</a></li>
   <li><a href="#">Tokyo</a></li>
 </ul>
</nav>

<article>
 <h1>London</h1>
```

London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.
Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium. </article>

<footer>Copyright © W3Schools.com</footer>

</body>
</html>

HTML Forms

The HTML **<form>** element defines a form that is used to collect user input:

<form>
.
form elements
.
</form> an HTML form contains **form elements**.

Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more.

The <input> Element

The **<input>** element is the most important form element.

The <input> element can be displayed in several ways, depending on the **type** attribute. Here are some examples:

Type	Description
<input type="text">	Defines a one-line text input field
<input type="radio">	Defines a radio button (for selecting one of many choices)
<input type="submit">	Defines a submit button (for submitting the form)

Text Input

<input type="text"> defines a one-line input field for **text input**:

Example:

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

<form>
  First name:<br>
```

```
  <input type="text" name="firstname">
  <br>
  Last name:<br>
  <input type="text" name="lastname">
</form>

<p>Note that the form itself is not visible.</p>

<p>Also note that the default width of a text input field is 20 characters.</p>

</body>
</html>
```

This is how it will look like in a browser: First name:

Last name:

Note: The form itself is not visible. Also note that the default width of a text field is 20 characters.

Radio Button Input

<input type="radio"> defines a **radio button**.

Radio buttons let a user select ONE of a limited number of choices:

Example

```
<form>
  <input type="radio" name="gender" value="male" checked> Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
  <input type="radio" name="gender" value="other"> Other
</form>
```

This is how the HTML code above will be displayed in a browser:

- (•) Male
- ( ) Female
- ( ) Other

The Submit Button

<input type="submit"> defines a button for **submitting** the form data to a **form-handler**.

The form-handler is typically a server page with a script for processing input data. The form-handler is specified in the form's **action** attribute:

Example

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
```

</form>
This is how the HTML code above will be displayed in a browser: First name:

Mickey

Last name:

Mouse

Submit

Input Type Reset

<input type="reset"> defines a **reset button** that will reset all form values to their default values:

Example

```
<form action="/action_page.php">
  First name:<br>
  <input    type="text"    name="firstname"    value="Mickey"><br>
Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
  <input type="reset">
</form>
```

input Type Checkbox

<input type="checkbox"> defines a **checkbox**.

Checkboxes let a user select ZERO or MORE options of a limited number of choices. **Example**

```
<form>
  <input type="checkbox" name="vehicle1" value="Bike"> I have a bike<br>   <input
type="checkbox" name="vehicle2" value="Car"> I have a car
</form>
```

This is how the HTML code above will be displayed in a browser:

I have a bike

I have a car

HTML5 Input Types

HTML5 added several new input types:

- color
- date
- datetime-local
- email
- month
- number
- range
- search

- tel
- time
- url
- week

New input types that are not supported by older web browsers, will behave as <input type="text">.

Input Type Color

The **<input type="color">** is used for input fields that should contain a color. Depending on browser support, a color picker can show up in the input field.

Example

```
<form>
  Select your favorite color:
  <input type="color" name="favcolor">
</form>
```

Input Type Date

The **<input type="date">** is used for input fields that should contain a date. Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  Birthday:
  <input type="date" name="bday">
</form>
```

You can also add restrictions to dates:

Example

```
<form>
  Enter a date before 1980-01-01:
  <input type="date" name="bday" max="1979-12-31"><br>
Enter a date after 2000-01-01:
  <input type="date" name="bday" min="2000-01-02"><br> </form>
```

Input Type Datetime-local

The **<input type="datetime-local">** specifies a date and time input field, with no time zone. Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  Birthday (date and time):
  <input type="datetime-local" name="bdaytime"> </form>
```

Input Type Email

The **<input type="email">** is used for input fields that should contain an e-mail address. Depending on browser support, the e-mail address can be automatically validated when submitted.

Some smartphones recognize the email type, and adds ".com" to the keyboard to match email input.

Example <form>
  E-mail:
  <input type="email" name="email">
</form>

Input Type Month
The **<input type="month">** allows the user to select a month and year.
Depending on browser support, a date picker can show up in the input field.

Example
<form>
  Birthday (month and year):
  <input type="month" name="bdaymonth">
</form>

Input Type Number
The **<input type="number">** defines a **numeric** input field.
You can also set restrictions on what numbers are accepted.
The following example displays a numeric input field, where you can enter a value from 1 to 5:

Example
<form>
  Quantity (between 1 and 5):
  <input type="number" name="quantity" min="1" max="5">
</form>

The <select> Element
The **<select>** element defines a **drop-down list**:
Example
<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
The **<option>** elements defines an option that can be selected.
By default, the first item in the drop-down list is selected.
To define a pre-selected option, add the **selected** attribute to the option:
Allow Multiple Selections:
Use the **multiple** attribute to allow the user to select more than one value:
Example
<select name="cars" size="4" **multiple**>

```
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select
```

The <textarea> Element

The **<textarea>** element defines a multi-line input field (**a text area**):

Example

```
<textarea name="message" rows="10" cols="30"> The
cat was playing in the garden.
</textarea>
```

The **rows** attribute specifies the visible number of lines in a text area.

The **cols** attribute specifies the visible width of a text area.

This is how the HTML code above will be displayed in a browser:

You can also define the size of the text area by using CSS:

Example

```
<textarea name="message" style="width:200px; height:600px"> The
cat was playing in the garden.
</textarea>
```

The <button> Element

The **<button>** element defines a clickable **button**:

Example

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

HTML5 Form Elements

HTML5 added the following form elements:

- <datalist>
- <output>

Note: Browsers do not display unknown elements. New elements that are not supported in older browsers will not "destroy" your web page.

HTML5 <datalist> Element

The **<datalist>** element specifies a list of pre-defined options for an <input> element.

Users will see a drop-down list of the pre-defined options as they input data.

The **list** attribute of the <input> element, must refer to the **id** attribute of the <datalist> element.

Example

```
<form action="/action_page.php">
  <input list="browsers">
  <datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
```

```
  </datalist>
</form>
```

HTML5 <output> Element

The <output> element represents the result of a calculation (like one performed by a script). **Example** Perform a calculation and show the result in an <output> element:

```
<form action="/action_page.php"
  oninput="x.value=parseInt(a.value)+parseInt(b.value)">
  0
  <input type="range"  id="a" name="a" value="50">  100
+
  <input type="number" id="b" name="b" value="50">
  =
  <output name="x" for="a b"></output>
  <br><br>
  <input type="submit">
</form>
```

HTML <frame> Tag.

Example

A simple three-framed page:

```
<frameset cols="25%,50%,25%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
  <frame src="frame_c.htm">
</frameset>
```

Frameset tag is not supported by html5.

CSS

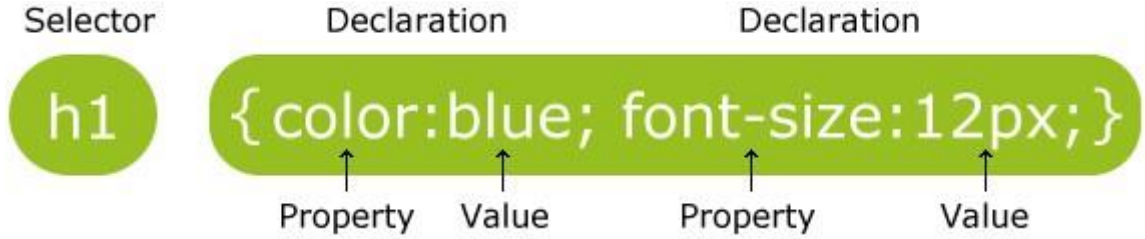
- **CSS** stands for **C**ascading **S**tyle **S**heets
- CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
- CSS **saves a lot of work**. It can control the layout of multiple web pages all at once

External stylesheets are stored in **CSS files Why Use CSS?**

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

CSS Syntax

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



The selector points to the HTML element you want to style.
The declaration block contains one or more declarations separated by semicolons.
Each declaration includes a CSS property name and a value, separated by a colon.
A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.
In the following example all <p> elements will be center-aligned, with a red text color: Example

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

CSS Selectors

CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more.

The element Selector

The element selector selects elements based on the element name.
You can select all <p> elements on a page like this (in this case, all <p> elements will be centeraligned, with a red text color):
Example

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

The id Selector

The id selector uses the id attribute of an HTML element to select a specific element.
The id of an element should be unique within a page, so the id selector is used to select one unique element!
To select an element with a specific id, write a hash (#) character, followed by the id of the element.
The style rule below will be applied to the HTML element with id="para1":
Example #para1

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

The class Selector

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

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

You can also specify that only specific HTML elements should be affected by a class. In the example below, only <p> elements with class="center" will be center-aligned:

Example

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

Grouping Selectors

If you have elements with the same style definitions, like this:

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

h2 {
   text-align:          center;
color: red;
}

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

It will be better to group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

In the example below we have grouped the selectors from the code above:

Example

```
h1, h2, p {     text-
align: center;
color: red;
}
```

CSS Comments

Comments are used to explain the code, and may help when you edit the source code at a later date. Comments are ignored by browsers.

A CSS comment starts with /* and ends with */. Comments can also span multiple lines: Example

```
p {
   color: red;
   /* This is a single-line comment */
   text-align: center;
}

/* This is a
multi-line
comment */
```

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet Inline style

External Style Sheet

With an external style sheet, you can change the look of an entire website by changing just one file! Each page must include a reference to the external style sheet file inside the <link> element. The <link> element goes inside the <head> section:

Example

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

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a .css extension.

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

```
{
   background-color: lightblue;
}

h1 {     color: navy;
margin-left: 20px; }
```

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

Internal Style Sheet

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

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

Example

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

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

Inline Styles

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

The example below shows how to change the color and the left margin of a <h1> element: **Example**

```
<h1 style="color:blue;margin-left:30px;">This is a heading</h1>
```

Tip: An inline style loses many of the advantages of a style sheet (by mixing content with presentation).

Cascading Order

What style will be used when there is more than one style specified for an HTML element?

Generally speaking we can say that all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section) 3. Browser default

So, an inline style (inside a specific HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or a browser default value.

CSS Colors

Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

Color Names

In HTML, a color can be specified by using a color name:

- Tomato
- Orange
- DodgerBlue
- MediumSeaGreen
- Gray
- SlateBlue
- Violet
- LightGray

Example

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

Text Color

Example

```
<h1 style="color:Tomato;">Hello World</h1>
```

Lorem ipsum...
Ut wisi enim...

Border Color

You can set the color of borders:

Example

<h1 style="border:2px solid Tomato;">Hello World</h1>
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
<h1 style="border:2px solid Violet;">Hello World</h1>

Color Values

In HTML, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

Same as color name "Tomato":

Example

<h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>

RGB Value

In HTML, a color can be specified as an RGB value, using this formula:

rgb(*red, green, blue*)

Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255. For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255) and the others are set to 0.

To display the color black, all color parameters must be set to 0, like this: rgb(0, 0, 0).

To display the color white, all color parameters must be set to 255, like this: rgb(255, 255, 255).

HEX Value

In HTML, a color can be specified using a hexadecimal value in the form: ***#rrggbb***

Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).

For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).

Example

#ff0000
#0000ff
#3cb371
#ee82ee
#ffa500
#6a5acd

HSL Value

In HTML, a color can be specified using hue, saturation, and lightness (HSL) in the form:

hsl(*hue, saturation, lightness*)

Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue. Saturation is a percentage value, 0% means a shade of gray, and 100% is the full color.
Lightness is also a percentage, 0% is black, 50% is neither light or dark, 100% is white
Example **hsl(0,**
100%, 50%) hsl(240,
100%, 50%) hsl(147,
50%, 47%) hsl(300,
76%, 72%) hsl(39,
100%, 50%) hsl(248,
53%, 58%)

Saturation

Saturation can be describe as the intensity of a color.
100% is pure color, no shades of gray
50% is 50% gray, but you can still see the color.
0% is completely gray, you can no longer see the color.
Example **hsl(0,**
100%, 50%) hsl(0,
80%, 50%) hsl(0,
60%, 50%) hsl(0,
40%, 50%) hsl(0,
20%, 50%) hsl(0,
0%, 50%)

RGBA Value

RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color.
An RGBA color value is specified with: **rgba(*red,***
***green, blue, alpha*)**
The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):
Example **rgba(255,**
99, 71, 0) rgba(255,
99, 71, 0.2) rgba(255,
99, 71, 0.4) rgba(255,
99, 71, 0.6) rgba(255,
99, 71, 0.8)
rgba(255, 99, 71, 1)

HSLA Value

HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity for a color.
An HSLA color value is specified with: **hsla(*hue,***
***saturation, lightness, alpha*)**
The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):
Example **hsla(9, 100%,**
64%, 0) hsla(9, 100%,
64%, 0.2) hsla(9, 100%,

64%, 0.4) hsla(9, 100%,
64%, 0.6) hsla(9, 100%,
64%, 0.8)
hsla(9, 100%, 64%, 1)

CSS Backgrounds

The CSS background properties are used to define the background effects for elements.

CSS background properties: □ background-color

- background-image
- background-repeat
- background-attachment
- background-position

Background Color

The background-color property specifies the background color of an element.

The background color of a page is set like this: Example body {

```
  background-color: lightblue;
}
```

Background Image

The background-image property specifies an image to use as the background of an element.

By default, the image is repeated so it covers the entire element.

The background image for a page can be set like this:

Example body {

```
  background-image: url("paper.gif");
}
```

Background Image - Repeat Horizontally or Vertically

By default, the background-image property repeats an image both horizontally and vertically.

Some images should be repeated only horizontally or vertically, or they will look strange, like this:

Example body {

```
  background-image: url("gradient_bg.png");
}
```

If the image above is repeated only horizontally (background-repeat: repeat-x;), the background will look better:

Example body

{

```
  background-image:  url("gradient_bg.png");          background-
repeat: repeat-x;
}
```

Tip: To repeat an image vertically, set background-repeat: repeat-y;

Background Image - Set position and no-repeat

Showing the background image only once is also specified by the background-repeat property:

Example body {

```
  background-image:  url("img_tree.png");          background-
repeat: no-repeat;
```

} in the example above, the background image is shown in the same place as the text. We want to change the position of the image, so that it does not disturb the text too much. The position of the image is specified by the background-position property: Example body {
background-image: url("img_tree.png");
background-repeat: no-repeat;
background-position: right top;
}

Background Image - Fixed position

To specify that the background image should be fixed (will not scroll with the rest of the page), use the background-attachment property: Example body {
background-image: url("img_tree.png");
background-repeat: no-repeat;
background-position: right top;
background-attachment: fixed;
}

Border Style

The border-style property specifies what kind of border to display.
The following values are allowed:

- dotted - Defines a dotted border
- dashed - Defines a dashed border
- solid - Defines a solid border
- double - Defines a double border
- groove - Defines a 3D grooved border. The effect depends on the border-color value
- ridge - Defines a 3D ridged border. The effect depends on the border-color value
- inset - Defines a 3D inset border. The effect depends on the border-color value
- outset - Defines a 3D outset border. The effect depends on the border-color value
- none - Defines no border
- hidden - Defines a hidden border

The border-style property can have from one to four values (for the top border, right border, bottom border, and the left border).

Example

```
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}
p.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style: dotted dashed solid double;} Result:
```

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border. The effect depends on the border-color value.

A ridge border. The effect depends on the border-color value.

An inset border. The effect depends on the border-color value.

An outset border. The effect depends on the border-color value.

No border.

A hidden border.

A mixed border.

Border Width

The border-width property specifies the width of the four borders.

The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three predefined values: thin, medium, or thick.

The border-width property can have from one to four values (for the top border, right border, bottom border, and the left border).

5px border-width

Example p.one {

border-style: solid; border-width: 5px;

}

p.two {

border-style: solid;

border-width: medium;

}

p.three { border-style: solid;

border-width: 2px 10px 4px 20px;

}

Border Color

The border-color property is used to set the color of the four borders.

The color can be set by:

- name - specify a color name, like "red"
- Hex - specify a hex value, like "#ff0000"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- transparent

The border-color property can have from one to four values (for the top border, right border, bottom border, and the left border).
If border-color is not set, it inherits the color of the element.
Red border
Example p.one
{
  border-style: solid; border-
color: red;
}

p.two {
  border-style: solid; border-
color: green;
}

p.three { border-
style: solid;
  border-color: red green blue yellow; }

Border - Individual Sides

From the examples above you have seen that it is possible to specify a different border for each side. In CSS, there are also properties for specifying each of the borders (top, right, bottom, and left): Different Border Styles
Example
p { border-top-style: dotted;
border-right-style: solid;
border-bottom-style: dotted;
border-left-style: solid;
}

Border - Shorthand Property

As you can see from the examples above, there are many properties to consider when dealing with borders.
To shorten the code, it is also possible to specify all the individual border properties in one property. The border property is a shorthand property for the following individual border properties:

- border-width
- border-style (required)
- border-color Example

p {
  border: 5px solid red;
}

Rounded Borders

The border-radius property is used to add rounded borders to an element: Example
p { border: 2px solid
red; border-radius:
5px; }

CSS Margins

This element has a margin of 70px.

The CSS margin properties are used to create space around elements, outside of any defined borders. With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

Margin - Individual Sides

CSS has properties for specifying the margin for each side of an element:

- margin-top
- margin-right
- margin-bottom
- margin-left

All the margin properties can have the following values:

- auto - the browser calculates the margin
- *length* - specifies a margin in px, pt, cm, etc.
- *%* - specifies a margin in % of the width of the containing element
- inherit - specifies that the margin should be inherited from the parent element **Tip:** Negative values are allowed.

The following example sets different margins for all four sides of a <p> element: Example

```
p {
  margin-top: 100px;  margin-
bottom: 100px;  margin-right:
150px;  margin-left: 80px;
}
```

Margin - Shorthand Property

- **margin: 25px 50px 75px 100px;**
  - top margin is 25px
  - right margin is 50px
  - bottom margin is 75px
  - left margin is 100px

Example

```
p {
  margin: 25px 50px 75px 100px;
}
```

The auto Value

You can set the margin property to auto to horizontally center the element within its container. The element will then take up the specified width, and the remaining space will be split equally between the left and right margins: Example

```
div {  width: 300px;
margin: auto;  border:
1px solid red;
}
```

The inherit Value

This example lets the left margin of the <p class="ex1"> element be inherited from the parent element ( ): Example

```
div {    border: 1px solid
red;          margin-left:
100px;
}

p.ex1 {     margin-left:
inherit;
}
```

CSS Padding

The CSS padding properties are used to generate space around an element's content, inside of any defined borders.

With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

Padding - Individual Sides

CSS has properties for specifying the padding for each side of an element:

- padding-top
- padding-right
- padding-bottom
- padding-left

All the padding properties can have the following values:

- *length* - specifies a padding in px, pt, cm, etc.
- *%* - specifies a padding in % of the width of the containing element
- inherit - specifies that the padding should be inherited from the parent element **Note:** Negative values are not allowed.

The following example sets different padding for all four sides of a element: Example

```
div  {          padding-top:
50px;       padding-right:
30px;    padding-bottom:
50px;          padding-left:
80px; }
```

Padding - Shorthand Property

- **padding: 25px 50px 75px 100px;**
  - top padding is 25px
  - right padding is 50px
  - bottom padding is 75px
  - left padding is 100px

Example

```
div {
  padding: 25px 50px 75px 100px;
}
```

Setting height and width

The height and width properties are used to set the height and width of an element.

The height and width can be set to auto (this is default. Means that the browser calculates the height and width), or be specified in *length values*, like px, cm, etc., or in percent (%) of the containing block.

This element has a height of 200 pixels and a width of 50%

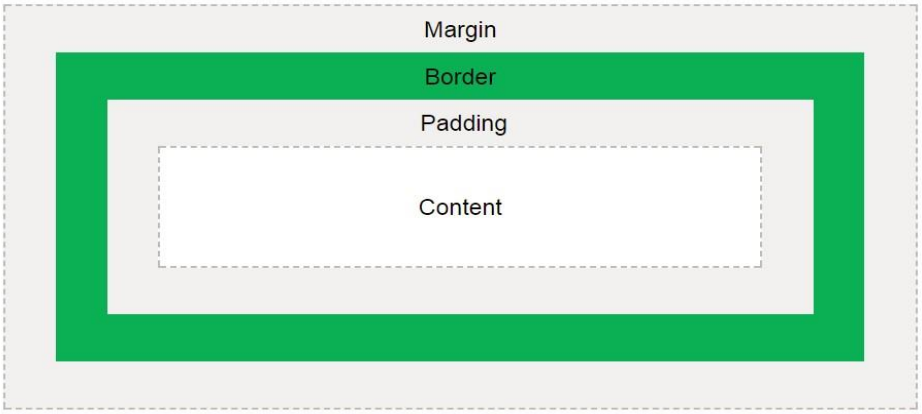
Example

```
div {       height:
200px;      width:
50%;
   background-color: powderblue;
}
```

The CSS Box Model

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

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



Explanation of the different parts:

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

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

Example

```
div {
   width: 300px;     border:
25px solid green;
padding: 25px;     margin:
25px;
}
```

Text Transformation

The text-transform property is used to specify uppercase and lowercase letters in a text.

It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word:

Example
```
p.uppercase {
text-transform: uppercase;
}

p.lowercase {
text-transform: lowercase;
}

p.capitalize {
text-transform: capitalize;
}
```
Letter Spacing

The letter-spacing property is used to specify the space between the characters in a text.

The following example demonstrates how to increase or decrease the space between characters:

Example
```
h1 {
  letter-spacing: 3px;
}

h2 {
  letter-spacing: -3px;
}
```
Line Height

The line-height property is used to specify the space between lines:

Example
```
p.small {
  line-height: 0.8;
}

p.big {
line-height: 1.8;
}
```
Word Spacing

The word-spacing property is used to specify the space between the words in a text.

The following example demonstrates how to increase or decrease the space between words:

Example
```
h1 {
  word-spacing: 10px;
}

h2 {
  word-spacing: -5px;
}
```
Text Shadow

The text-shadow property adds shadow to text.

The following example specifies the position of the horizontal shadow (3px), the position of the vertical shadow (2px) and the color of the shadow (red): Example h1 {

```
  text-shadow: 3px 2px red;
}
```

Font Family

The font family of a text is set with the font-family property.

The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font, and so on.

Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

Note: If the name of a font family is more than one word, it must be in quotation marks, like: "Times New Roman".

More than one font family is specified in a comma-separated list: Example

```
p {
  font-family: "Times New Roman", Times, serif;
}
```

Font Style

The font-style property is mostly used to specify italic text.

This property has three values:

- normal - The text is shown normally
- italic - The text is shown in italics
- oblique - The text is "leaning" (oblique is very similar to italic, but less supported)

Example

```
p.normal {
  font-style: normal;
}

p.italic {
  font-style: italic;
}

p.oblique {
  font-style: oblique;
}
```

Set Font Size With Pixels

Setting the text size with pixels gives you full control over the text size:

Example h1 {

```
  font-size: 40px;
}

h2 {
  font-size: 30px;
}
```

```
p {
   font-size: 14px; }
```

CSS Icons

How To Add Icons

The simplest way to add an icon to your HTML page, is with an icon library, such as Font Awesome. Add the name of the specified icon class to any inline HTML element (like <i> or). All the icons in the icon libraries below, are scalable vectors that can be customized with CSS (size, color, shadow, etc.)

Font Awesome Icons

To use the Font Awesome icons, add the following line inside the <head> section of your HTML page:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/fontawesome.min.css">
```

Note: No downloading or installation is required!

Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/fontawesome.min.css">
</head>
<body>

<i class="fa fa-cloud"></i>
<i class="fa fa-heart"></i>
<i class="fa fa-car"></i>
<i class="fa fa-file"></i>
<i class="fa fa-bars"></i>

</body>
</html>
```

Bootstrap Icons

To use the Bootstrap glyphicons, add the following line inside the <head> section of your HTML page:

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
```

Note:
No downloading or installation is required!

Example

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

```
<head>
<link rel="stylesheet"href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
>
</head>
<body>

<i class="glyphicon glyphicon-cloud"></i>
<i class="glyphicon glyphicon-remove"></i>
<i class="glyphicon glyphicon-user"></i>
<i class="glyphicon glyphicon-envelope"></i>
<i class="glyphicon glyphicon-thumbs-up"></i>

</body>
</html>
```

Styling Links

Links can be styled with any CSS property (e.g. color, font-family, background, etc.).

In addition, links can be styled differently depending on what **state** they are in.
The four links states are:

- a:link - a normal, unvisited link
- a:visited - a link the user has visited
- a:hover - a link when the user mouses over it
- a:active - a link the moment it is clicked

Example

```
/* unvisited link */
a:link {     color:
red;
}

/* visited link */
a:visited {
color: green;
}

/* mouse over link */
a:hover {     color:
hotpink;
}

/* selected link */
a:active {
color: blue;
```

```
}
```

Background Color

The background-color property can be used to specify a background color for links: Example

```
a:link {
   background-color: yellow;
}

a:visited {
   background-color: cyan;
}

a:hover {
   background-color: lightgreen;
}

a:active {
   background-color: hotpink; }
```

CSS List Properties

The CSS list properties allow you to:

- Set different list item markers for ordered lists
- Set different list item markers for unordered lists
- Set an image as the list item marker
- Add background colors to lists and list items

Different List Item Markers

The list-style-type property specifies the type of list item marker.

The following example shows some of the available list item markers:

```
<!DOCTYPE html>
<html>
<head>
<style> ul.a
{
   list-style-type: circle;
}

ul.b {
   list-style-type: square;
}

ol.c {
   list-style-type: upper-roman;
}
```

```
ol.d {
   list-style-type: lower-alpha;
}
</style>
</head>
<body>

<p>Example of unordered lists:</p>
<ul class="a">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ul>

<ul class="b">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ul>

<p>Example of ordered lists:</p>
<ol class="c">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>

<ol class="d">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>

</body>
</html>
```

An Image as The List Item Marker

The list-style-image property specifies an image as the list item marker: Example

```
ul {
   list-style-image: url('sqpurple.gif');
}
```

Remove Default Settings

The list-style-type:none property can also be used to remove the markers/bullets. Note that the list also has default margin and padding. To remove this, add margin:0 and padding:0 to <ul> or <ol>: Example

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

Styling List With Colors

We can also style lists with colors, to make them look a little more interesting.
Anything added to the <ol> or <ul> tag, affects the entire list, while properties added to the <li> tag will affect the individual list items: Example

```
ol {
   background:              #ff9999;
padding: 20px;
}

ul {
   background:              #3399ff;
padding: 20px;
}

ol li {
   background: #ffe5e5;
padding: 5px;     margin-
left: 35px;
}

ul li {
   background: #cce5ff;
   margin: 5px;
}
```

Table Borders

To specify table borders in CSS, use the border property.
The example below specifies a black border for <table>, <th>, and <td> elements:
Example

```
table, th, td {
  border: 1px solid black;
}
```

Collapse Table Borders

The border-collapse property sets whether the table borders should be collapsed into a single border: Example

```
table {
   border-collapse: collapse;
}
```

```
table, th, td {
  border: 1px solid black; }
```

XHTML

What Is XHTML?

- XHTML stands for E**X**tensible **H**yper**T**ext **M**arkup **L**anguage
- XHTML is almost identical to HTML
- XHTML is stricter than HTML
- XHTML is HTML defined as an XML application
- XHTML is supported by all major browsers

Why XHTML?

Many pages on the internet contain "bad" HTML.

This HTML code works fine in most browsers (even if it does not follow the HTML rules):

```
<html>
<head>
 <title>This is bad HTML</title>

<body>
 <h1>Bad HTML
 <p>This is a paragraph
</body>
```

Today's market consists of different browser technologies. Some browsers run on computers, and some browsers run on mobile phones or other small devices. Smaller devices often lack the resources or power to interpret "bad" markup.

XML is a markup language where documents must be marked up correctly (be "well-formed").

By combining the strengths of HTML and XML, XHTML was developed. XHTML is HTML redesigned as XML.

The Most Important Differences from HTML:

Document Structure

- XHTML DOCTYPE is **mandatory**
- The xmlns attribute in <html> is **mandatory**
- <html>, <head>, <title>, and <body> are **mandatory**

XHTML Elements

- XHTML elements must be **properly nested**
- XHTML elements must always be **closed**
- XHTML elements must be in **lowercase**
- XHTML documents must have **one root element**

XHTML Attributes

- Attribute names must be in **lower case**
- Attribute values must be **quoted**
- Attribute minimization is **forbidden**

<!DOCTYPE ....> Is Mandatory

An XHTML document must have an XHTML DOCTYPE declaration.
A complete list of all the XHTML Doctypes is found in our HTML Tags Reference.
The <html>, <head>, <title>, and <body> elements must also be present, and the xmlns attribute in <html> must specify the xml namespace for the document.
This example shows an XHTML document with a minimum of required tags:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Title of document</title>
</head>

<body>
  some content
</body>

</html>
```

XHTML Elements Must Be Properly Nested
In HTML, some elements can be improperly nested within each other, like this:
This text is bold and italic
In XHTML, all elements must be properly nested within each other, like this:
This text is bold and italic **XHTML**
Elements Must Always Be Closed
This is wrong:
This is a paragraph
This is another paragraph
This is correct:
This is a paragraph
This is another paragraph

Empty Elements Must Also Be Closed This
is wrong:
A break:
A horizontal rule: <hr>
An image: <img src="happy.gif" alt="Happy face"> This
is correct:
A break:
A horizontal rule: <hr />
An image:
XHTML Elements Must Be In Lower Case
This is wrong:

<BODY>
<P>This is a paragraph</P>
</BODY>
This is correct:
<body>
This is a paragraph
</body>
XHTML Attribute Names Must Be In Lower Case This
is wrong:
<table WIDTH="100%">
This is correct:
<table width="100%">
Attribute Values Must Be Quoted
This is wrong: <table width=100%>
This is correct:
<table width="100%">
Attribute Minimization Is Forbidden Wrong:
<input type="checkbox" name="vehicle" value="car" checked /> Correct:
<input type="checkbox" name="vehicle" value="car" checked="checked" /> Wrong:
<input type="text" name="lastname" disabled /> Correct:
<input type="text" name="lastname" disabled="disabled" />

How to Convert from HTML to XHTML

1. Add an XHTML <!DOCTYPE> to the first line of every page
2. Add an xmlns attribute to the html element of every page
3. Change all element names to lowercase
4. Close all empty elements
5. Change all attribute names to lowercase
6. Quote all attribute values

UNIT II

Java Script: An introduction to JavaScript–JavaScript DOM Model-Date and Objects,-Regular Expressions- Exception Handling-Validation-Built-in objects-Event Handling-DHTML with JavaScript. Servlets: Java Servlet Architecture- Servlet Life Cycle- Form GET and POST actions- Session Handling- Understanding Cookies.
Installing and Configuring Apache Tomcat Web Server;- DATABASE CONNECTIVITY: JDBC perspectives, JDBC program example - JSP: Understanding Java Server Pages-JSP Standard Tag Library(JSTL)-Creating HTML forms by embedding JSP code.

An introduction to JavaScript:

JavaScript is a lightweight, interpreted programming language. It is designed for creating networkcentric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java.

Advantages of JavaScript

The merits of using JavaScript are −

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors. **Limitations of JavaScript**

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

Java script syntax:

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>**.

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

The script tag takes two important attributes −

- **Language** − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type** − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript". So your JavaScript segment will look like −

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

  JavaScript code

</script>
```

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus −

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->. Example:

```
<html>
  <body>
    <script language="javascript" type="text/javascript">          document.write("Hello World!")
     </script>
  </body>
</html>
```

➢ Javascript can be used in both head and body sections.

JavaScript Variables

Variables are declared with the **var** keyword as follows.

```
<script type="text/javascript">
  <!--      var
money;
var name;
  //-->
</script>
```

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables** − A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<html>
```

```
  <body onload = checkscope();>
    <script type = "text/javascript">
      <!--
```

```
        var myVar = "global"; // Declare a global variable          function checkscope( ) {
          var myVar = "local";  // Declare a local variable
document.write(myVar);
        }
      //-->
    </script>
  </body>
</html>
```

JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while

do	import	static	with
double	in	super	

JavaScript Operators

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

+,-,*,/,%,++,--

Example

The following code shows how to use arithmetic operators in JavaScript.

```
<html>
  <body>
      <script type="text/javascript">
      <!--          var
a = 33;          var b
= 10;          var c =
"Test";
          var   linebreak   =   "<br   />";
document.write("a + b = ");
          result = a + b;
document.write(result);
document.write(linebreak);
document.write("a - b = ");
result = a - b;
          document.write(result);
document.write(linebreak);
document.write("a / b = ");
result = a / b;
          document.write(result);
document.write(linebreak);
document.write("a % b = ");
          result = a % b;
document.write(result);
document.write(linebreak);
document.write("a + b + c = ");
          result = a + b + c;
document.write(result);
document.write(linebreak);
a = ++a;
          document.write("++a = ");
result               =               ++a;
document.write(result);
          document.write(linebreak);
b = --b;
          document.write("--b     =     ");
result = --b;
          document.write(result);
          document.write(linebreak);
      //-->
    </script>
    Set the variables to different values and then try...
  </body> </html>
```

Output

a + b = 43
a - b = 23 a
/ b = 3.3 a
% b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...

Comparison Operators

==,!=,<,<=,>,>=

Logical Operators

JavaScript supports the following logical operators − &&,||,!

Bitwise Operators

JavaScript supports the following bitwise operators − ✓

& (Bitwise AND)

- ✓ | (BitWise OR)
- ✓ ^ (Bitwise XOR)
- ✓ ~ (Bitwise Not)
- ✓ << (Left Shift)
- ✓ >> (Right Shift)
- ✓ >>> (Right shift with Zero)

This operator is just like the >> operator, except that the bits shifted in on the left are always zero. **Ex:** (A >>> 1) is 1.

Assignment Operators

- ✓ **= (Simple Assignment)**
- ✓ **+= (Add and Assignment)**
- ✓ **−= (Subtract and Assignment)** ✓ ***= (Multiply and Assignment)**
- ✓ **/= (Divide and Assignment)**
- ✓ **%= (Modules and Assignment)**

Conditional Operator (? :)

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

JavaScript - if...else Statement

JavaScript supports conditional statements which are used to perform different actions based on different conditions.

JavaScript supports the following forms of **if..else** statement −

- if statement
- if...else statement
- if...else if... statement.

if statement

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

The syntax for a basic if statement is as follows −

```
if (expression){
  Statement(s) to be executed if expression is true }
```

Example

Try the following example to understand how the **if** statement works.

```
<html>
  <body>
    <script type="text/javascript">
      <!--
        var age = 20;
if( age > 18 ){
          document.write("<b>Qualifies for driving</b>");
        }
      //-->
    </script>
        <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Qualifies for driving
Set the variable to different value and then try...
```

if...else statement:

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way. **Syntax**

```
if (expression){
  Statement(s) to be executed if expression is true
}
else{
  Statement(s) to be executed if expression is false
}
```

if...else if... statement

The **if...else if...** statement is an advanced form of **if…else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows −

```
if (expression 1){
   Statement(s) to be executed if expression 1 is true
}
else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}
else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}
else{
   Statement(s) to be executed if no expression is true
}
```

JavaScript - Switch Case

Syntax

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

```
switch (expression)
{
   case condition 1: statement(s)
break;
      case condition 2: statement(s)
break;
   ...

   case condition n: statement(s)
break;
      default: statement(s)
}
```

The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases. We will explain **break** statement in *Loop Control* chapter.

Example

Try the following example to implement switch-case statement.

```
<html>
   <body>
      <script type="text/javascript">
      <!--
         var grade='A';
         document.write("Entering switch block<br />");          switch (grade)
```

```
        {
          case 'A': document.write("Good job<br />");
```

```
            break;
                    case 'B': document.write("Pretty good<br />");
break;
                    case 'C': document.write("Passed<br />");
break;
                    case 'D': document.write("Not so good<br />");
break;

            case 'F': document.write("Failed<br />");
break;
                    default:  document.write("Unknown grade<br />")
          }
          document.write("Exiting switch block");
       //-->
    </script>

    <p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering switch block
Good job
Exiting switch block
Set the variable to different value and then try...
```

JavaScript - While Loops

Syntax

The syntax of **while loop** in JavaScript is as follows −

```
while (expression){
  Statement(s) to be executed if expression is true
}
```

Example

Try the following example to implement while loop.

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

```
      <!--          var count = 0;          document.write("Starting Loop ");
while (count < 10){
           document.write("Current Count : " + count + "<br />");
count++;
         }
               document.write("Loop stopped!");
      //-->
    </script>
        <p>Set the variable to different value and then try...</p>
  </body> </html>
```

Syntax

The syntax for **do-while** loop in JavaScript is as follows −

```
do{

  Statement(s) to be executed;

} while (expression);
```

Note − Don‘t miss the semicolon used at the end of the do...while loop.

JavaScript - For Loop

Syntax

The syntax of for loop is JavaScript is as follows −

```
for (initialization; test condition; iteration statement){
  Statement(s) to be executed if test condition is true
}
```

Example

Try the following example to learn how a for loop works in JavaScript.

```
<html>
  <body>
        <script type="text/javascript">
      <!--
var count;
        document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++){
document.write("Current Count : " + count );
document.write("<br />");
        }
              document.write("Loop stopped!");
      //-->
    </script>
        <p>Set the variable to different value and then try...</p>
</body>
</html>
```

JavaScript *for...in* loop

The **for...in** loop is used to loop through an object's properties. **Syntax**

```
for (variablename in object){
```

```
  statement or block to execute }
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

Example

Try the following example to implement _for-in' loop. It prints the web browser's **Navigator** object.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var aProperty;
        document.write("Navigator Object Properties<br /> ");
              for (aProperty in navigator) {          document.write(aProperty);
          document.write("<br />");
        }
        document.write ("Exiting from the loop!");
      //-->
    </script>
        <p>Set the variable to different object and then try...</p>
```

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

JavaScript – Functions

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Syntax

The basic syntax is shown here.

```
<script type="text/javascript">
  <!--
    function functionname(parameter-list)
    {
      statements
    }
  //-->
</script>
```

Example

Try the following example. It defines a function called sayHello that takes no parameters −

```
<script type="text/javascript">
<!--
    function sayHello()
    {
      alert("Hello there");
    }
  //-->
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
  <head>
      <script            type="text/javascript">
function sayHello()
      {
        document.write ("Hello there!");
      }
    </script>

  </head>
  <body>
    <p>Click the following button to call the function</p>
<form>
      <input type="button" onclick="sayHello()" value="Say Hello">
</form>
        <p>Use different text in write method and then try...</p>
</body>
</html>
```

Function Parameters

These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

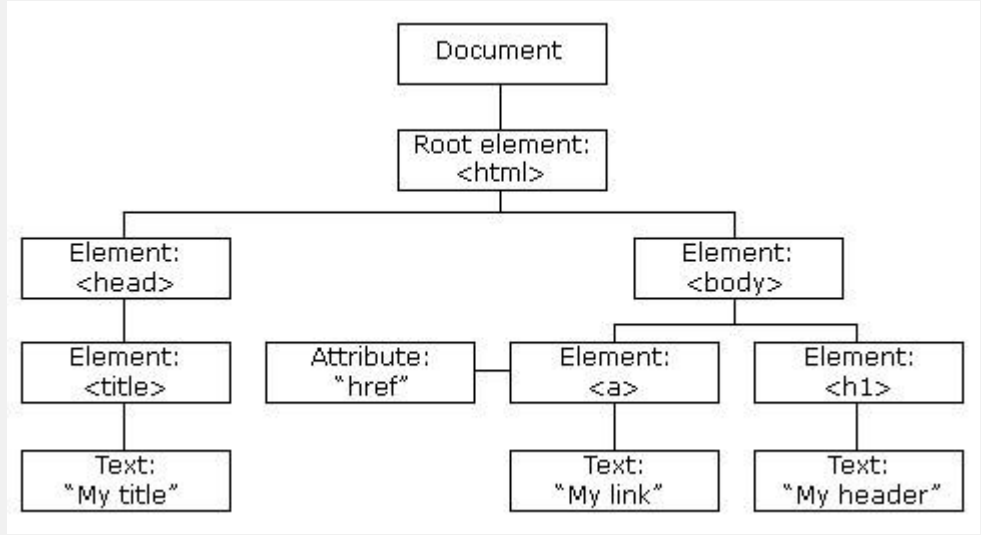
```
<html>
  <head>
      <script type="text/javascript">
      function sayHello(name, age)
      {
        document.write (name + " is " + age + " years old.");
      }
    </script>
     </head>
  <body>
    <p>Click the following button to call the function</p>          <form>
      <input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>
        <p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

2. JavaScript DOM Model

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

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



With the object model, JavaScript gets all the power it needs to create dynamic HTML:
The HTML DOM Tree of Objects

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

JavaScript - HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements).
HTML DOM properties are **values** (of HTML Elements) that you can set or change.
In the DOM, all HTML elements are defined as **objects**.
The programming interface is the properties and methods of each object.
A **property** is a value that you can get or set (like changing the content of an HTML element).
A **method** is an action you can do (like add or deleting an HTML element). **Example**
The following example changes the content (the innerHTML) of the <p> element with id="demo":
Example

```
<html>
<body>
<p id="demo"></p>
<script>
```

document.getElementById("demo").innerHTML = "Hello World!"; </script>
</body>
</html>

JavaScript HTML DOM Document

The HTML DOM document object is the owner of all other objects in your web page.

The HTML DOM Document Object

The document object represents your web page.

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

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

Finding HTML Elements

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

JavaScript HTML DOM Elements

This page teaches you how to find and access HTML elements in an HTML page.

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are a couple of ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

example <!DOCTYPE
html>
<html>
<body>
Hello World!
This example demonstrates the getElementById method!

<script>
var myElement = document.getElementById("intro");
document.getElementById("demo").innerHTML =
"The text from the intro paragraph is " + myElement.innerHTML; </script>
</body>
</html>

Finding HTML Elements by Tag Name This example finds all <p> elements:

Example

- var x = document.getElementsByTagName("p");

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use getElementsByClassName(). This example returns a list of all elements with class="intro".

Example

- var x = document.getElementsByClassName("intro");

JavaScript HTML DOM Events

HTML DOM allows JavaScript to react to HTML events:

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element. To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=*JavaScript*

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

3. **Date and Objects**

Date methods let you get and set date values (years, months, days, hours, minutes, seconds, milliseconds)

Date Get Methods

Get methods are used for getting a part of a date. Here are the most common (alphabetically):

Method	**Description**
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)

getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

The getTime() Method

getTime() returns the number of milliseconds since January 1, 1970:
example <!DOCTYPE
html>
<html>
<body>
The internal clock in JavaScript starts at midnight January 1, 1970.
The getTime() function returns the number of milliseconds since then:

<script> var d =
new Date();
document.getElementById("demo").innerHTML = d.getTime();
</script>
</body>
</html>
The getFullYear() Method getFullYear() returns the year of a

date as a four digit number:

Example
<script> var d =
new Date();
document.getElementById("demo").innerHTML = d.getFullYear(); </script>

The getDay() Method getDay() returns the

weekday as a number (0-6):

Example

```
<script> var d =
new Date();
document.getElementById("demo").innerHTML = d.getDay(); </script>
```

Date Set Methods

Set methods are used for setting a part of a date. Here are the most common (alphabetically):

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

The setFullYear() Method setFullYear() sets a date object to a specific date. In this example, to January 14, 2020:

Example

```
<script> var d =
new Date();
d.setFullYear(2020, 0, 14); document.getElementById("demo").innerHTML
= d;
</script>
```

Compare Dates

Dates can easily be compared.

The following example compares today's date with January 14, 2100:

Example

```
var today, someday, text; today = new Date(); someday = new Date();
someday.setFullYear(2100, 0, 14);

if (someday > today) {
   text = "Today is before January 14, 2100.";
} else {
   text = "Today is after January 14, 2100.";
}
document.getElementById("demo").innerHTML = text;
```

4. Regular Expressions

A regular expression is an object that describes a pattern of characters.
Regular expressions are used to perform pattern-matching and "search-and-replace" functions on text.

Syntax
/pattern/modifiers;

Modifiers

Modifiers are used to perform case-insensitive and global searches:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Brackets

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specified

Metacharacters

Metacharacters are characters with a special meaning:

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character

\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\uxxxx	Find the Unicode character specified by a hexadecimal number xxxx

Quantifiers

Quantifier	**Description**
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X* *n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string *n*

RegExp Object Properties

Property	Description
constructor	Returns the function that created the RegExp object's prototype
global	Checks whether the "g" modifier is set
ignoreCase	Checks whether the "i" modifier is set
lastIndex	Specifies the index at which to start the next match
multiline	Checks whether the "m" modifier is set
source	Returns the text of the RegExp pattern

RegExp Object Methods

Method	Description
compile()	Deprecated in version 1.5. Compiles a regular expression
exec()	Tests for a match in a string. Returns the first match
test()	Tests for a match in a string. Returns true or false
toString()	Returns the string value of the regular expression

5. Exception Handling

JavaScript Errors - Throw and Try to Catch

- The **try** statement lets you test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.
- The **finally** statement lets you execute code, after try and catch, regardless of the result.

When executing JavaScript code, different errors can occur. Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

Example:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script> try
{
   adddlert("Welcome guest!");
} catch(err)
{
```

```
   document.getElementById("demo").innerHTML = err.message; }
</script>
</body>
</html>
```

JavaScript try and catch

The **try** statement allows you to define a block of code to be tested for errors while it is being executed.
The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.
The JavaScript statements **try** and **catch** come in pairs:

```
try {
   Block of code to try
}
catch(err) {
   Block of code to handle errors
}
```

JavaScript Throws Errors

When an error occurs, JavaScript will normally stop and generate an error message.
The technical term for this is: JavaScript will **throw an exception (throw an error)**.

Input Validation Example

This example examines input. If the value is wrong, an exception (err) is thrown.
The exception (err) is caught by the catch statement and a custom error message is displayed:

```
<!DOCTYPE html>
<html>
<body>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="message"></p>

<script>
function    myFunction()    {
var message, x;
   message                =                document.getElementById("message");
message.innerHTML = "";
   x    =    document.getElementById("demo").value;
try {
       if(x == "") throw "empty";
if(isNaN(x)) throw "not a number";
x = Number(x);         if(x < 5) throw
"too low";
       if(x > 10) throw "too high";
   }
catch(err) {
```

```
    message.innerHTML = "Input is " + err;
  }
}
</script>
</body>
</html>
```

The finally Statement

The **finally** statement lets you execute code, after try and catch, regardless of the result:

```
try {
  Block of code to try
} catch(err)
{
  Block of code to handle errors
}  finally
{
  Block of code to be executed regardless of the try / catch result }
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="message"></p>
<script> function
myFunction() {    var
message, x;
  message = document.getElementById("message");
message.innerHTML    =    "";                    x    =
document.getElementById("demo").value;    try {
    if(x == "")  throw "is empty";
if(isNaN(x)) throw "is not a number";
x = Number(x);       if(x > 10)   throw "is
too high";
    if(x < 5)    throw "is too low";
  }
catch(err) {
    message.innerHTML = "Input " + err;
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
```

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

Error Name Values

Six different values can be returned by the error name property:

Error Name	**Description**
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred

6. Validation

JavaScript Form Validation

HTML form validation can be done by JavaScript.

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

JavaScript Example

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

HTML Form Example

```
<!DOCTYPE html>
<html>
<head> <script>
function validateForm() {
   var x = document.forms["myForm"]["fname"].value;
if (x == "") {
```

```
        alert("Name must be filled out");
        return false;
    }
}
</script>
</head>
<body>

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

Automatic HTML Form Validation

HTML form validation can be performed automatically by the browser:

```
<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
```

If we press submit button browser will automatically generate error.

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

Server side validation is performed by a web server, after input has been sent to the server. **Client side validation** is performed by a web browser, before input is sent to a web server.

HTML Constraint Validation

HTML5 introduced a new HTML validation concept called **constraint validation**.

HTML constraint validation is based on:

- Constraint validation **HTML Input Attributes**
- Constraint validation **CSS Pseudo Selectors**
- Constraint validation **DOM Properties and Methods**

Constraint Validation HTML Input Attributes

Attribute	Description
disabled	Specifies that the input element should be disabled

max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

7. Built-in objects in javascript

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the **new** keyword)
- Numbers can be objects (if defined with the **new** keyword)
- Strings can be objects (if defined with the **new** keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript defines 5 types of primitive data types:

- string
- number
- boolean
- null
- undefined
  - if x = 3.14, you can change the value of x. But you cannot change the value of 3.14.

Value	**Type**	**Comment**
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

Object Methods

Methods are **actions** that can be performed on objects.
Object properties can be both primitive values, other objects, and functions. An **object method** is an object property containing a **function definition**.

Property	**Value**
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

JavaScript objects are containers for named values, called properties and methods.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.
There are different ways to create new objects:

- Define and create a single object, using an object literal.
- Define and create a single object, with the keyword new.
- Define an object constructor, and then create objects of the constructed type.

Using an Object Literal

- This is the easiest way to create a JavaScript Object.
- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of name:value pairs (like age:50) inside curly braces {}.
- The following example creates a new JavaScript object with four properties

```
<!DOCTYPE html>
<html>
<body>
<p>Creating a JavaScript Object.</p>
<p id="demo"></p>
<script>
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
document.getElementById("demo").innerHTML = person.firstName
+ " is " + person.age + " years old.";
</script>
</body>
</html>
```

Using the JavaScript Keyword new

The following example also creates a new JavaScript object with four properties: Example:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script> var person = new
Object(); person.firstName
= "John"; person.lastName
= "Doe"; person.age = 50;
person.eyeColor = "blue";
document.getElementById("demo").innerHTML = person.firstName
+ " is " + person.age + " years old.";
</script>
</body>
</html>
```

JavaScript Properties

Properties are the values associated with a JavaScript object.

A JavaScript object is a collection of unordered properties.

Properties can usually be changed, added, and deleted, but some are read only.

Accessing JavaScript Properties

The syntax for accessing the property of an object is:

objectName.property // person.age or

objectName["*property*"] // person["age"]

or

objectName[*expression*] // x = "age"; person[x] Example:

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

<p>There are two different ways to access an object property:</p>
<p>You can use .property or ["property"].</p>

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

<script> var person
= {
firstname:"John",
lastname:"Doe",
age:50,
eyecolor:"blue"
};
document.getElementById("demo").innerHTML =
person.firstname + " is " + person.age + " years old.";
</script>
```

</body>
</html>

Adding New Properties

You can add new properties to an existing object by simply giving it a value.

Assume that the person object already exists - you can then give it new properties:

Example

person.nationality = "English";

Deleting Properties

The **delete** keyword deletes a property from an object:

Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}; delete person.age; // or delete person["age"];

The delete keyword deletes both the value of the property and the property itself. After deletion, the property cannot be used before it is added back again.

Property Attributes

All properties have a name. In addition they also have a value.

The value is one of the property's attributes. Other attributes are: enumerable, onfigurable, and writable. These attributes define how the property can be accessed .

In JavaScript, all attributes can be read, but only the value attribute can be changed (and only if the property is writable).

The this Keyword

In JavaScript, the thing called **this**, is the object that "owns" the JavaScript code.

The value of **this**, when used in a function, is the object that "owns" the function.

Note that **this** is not a variable. It is a keyword. You cannot change the value of **this**.

Accessing Object Methods

You access an object method with the following syntax:

objectName.methodName()

You will typically describe fullName() as a method of the person object, and fullName as a property.

The fullName property will execute (as a function) when it is invoked with ().

This example accesses the fullName() **method** of a person object:

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

<p>Creating and using an object method.</p>

<p>A method is actually a function definition stored as a property value.</p>

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

<script> var person = {
firstName: "John",
lastName : "Doe",     id
```

```
: 5566,     fullName :
function() {
      return this.firstName + " " + this.lastName;
   }
};

document.getElementById("demo").innerHTML = person.fullName(); </script>
</body>
</html>
```

Using Built-In Methods

This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

```
var message = "Hello world!"; var
x = message.toUpperCase();
```

The value of x, after execution of the code above will be:

HELLO WORLD!

Using an Object Constructor

The examples from the previous chapters are limited in many situations. They only create single objects.

Sometimes we like to have an "object type" that can be used to create many objects of one type. The standard way to create an "object type" is to use an object constructor function:

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

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

<script>
function Person(first, last, age, eye) {
this.firstName = first;
this.lastName = last;     this.age =
age;     this.eyeColor = eye;
}

var myFather = new Person("John", "Doe", 50, "blue"); var
myMother = new Person("Sally", "Rally", 48, "green");

document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ". My mother is " + myMother.age;
</script>

</body>
</html>
```

Built-in JavaScript Constructors

JavaScript has built-in constructors for native objects:

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

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

<script>
var x1 = new Object();   // A new Object object
var x2 = new String();   // A new String object var
x3 = new Number();   // A new Number object var
x4 = new Boolean();  // A new Boolean object var
x5 = new Array();    // A new Array object var x6
= new RegExp();   // A new RegExp object var x7
= new Function(); // A new Function object
var x8 = new Date();     // A new Date object

document.getElementById("demo").innerHTML =
"x1: " + typeof x1 + "<br>" +
"x2: " + typeof x2 + "<br>" +
"x3: " + typeof x3 + "<br>" +
"x4: " + typeof x4 + "<br>" +
"x5: " + typeof x5 + "<br>" +
"x6: " + typeof x6 + "<br>" +
"x7: " + typeof x7 + "<br>" +
"x8: " + typeof x8 + "<br>";
</script>

<p>There is no need to use String(), Number(), Boolean(), Array(), and RegExp()</p> <p>Read
the JavaScript tutorials.</p>

</body>
</html>
```

Output: x1:
object x2:
object x3:
object x4:
object x5:
object x6:
object x7:
function x8:
object
There is no need to use String(), Number(), Boolean(), Array(), and RegExp() Read
the JavaScript tutorials.

String Objects

Normally, strings are created as primitives: **var firstName = "John"**

But strings can also be created as objects using the **new** keyword: **var firstName = new String("John")**

Learn why strings should not be created as object in the chapter JS Strings.

Number Objects

Normally, numbers are created as primitives: **var x = 123**

But numbers can also be created as objects using the **new** keyword: **var x = new Number(123)** Learn why numbers should not be created as object in the chapter JS Numbers.

Boolean Objects

Normally, booleans are created as primitives: **var x = false**

But booleans can also be created as objects using the **new** keyword: **var x = new Boolean(false)**

Learn why booleans should not be created as object in the chapter JS Booleans.

8. Event Handling

JavaScript - Events

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

Try the following example.

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

```
      <!--
        function sayHello() {
          alert("Hello World")
        }
      //-->
    </script>
      </head>
    <body>
    <p>Click the following button and see result</p>
        <form>
```

```
    <input type="button" onclick="sayHello()" value="Say Hello" />
  </form>
   </body> </html>
```

onsubmit Event type onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

```
<html>
  <head>
     <script type="text/javascript">
     <!--
       function over() {
         document.write ("Mouse Over");
       }
            function out() {
         document.write ("Mouse Out");
       }
          //-->
   </script>
     </head>
  <body>
    <p>Bring your mouse inside the division to see the result:</p>

    <div onmouseover="over()" onmouseout="out()">
      <h2> This is inside the division </h2>
    </div>
     </body>
</html>
```

HTML 5 Standard Events

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

Attribute	Value	Description
Offline	script	Triggers when the document goes offline
Onabort	script	Triggers on an abort event
onafterprint	script	Triggers after the document is printed

onbeforeonload	script	Triggers before the document loads
onbeforeprint	script	Triggers before the document is printed
onblur	script	Triggers when the window loses focus
oncanplay	script	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	script	Triggers when media can be played to the end, without stopping for buffering
onchange	script	Triggers when an element changes
onclick	script	Triggers on a mouse click
oncontextmenu	script	Triggers when a context menu is triggered
ondblclick	script	Triggers on a mouse double-click
ondrag	script	Triggers when an element is dragged
ondragend	script	Triggers at the end of a drag operation
ondragenter	script	Triggers when an element has been dragged to a valid drop target
ondragleave	script	Triggers when an element is being dragged over a valid drop target
ondragover	script	Triggers at the start of a drag operation
ondragstart	script	Triggers at the start of a drag operation
ondrop	script	Triggers when dragged element is being dropped
ondurationchange	script	Triggers when the length of the media is changed
onemptied	script	Triggers when a media resource element suddenly becomes empty.
onended	script	Triggers when media has reach the end
onerror	script	Triggers when an error occur

onfocus	script	Triggers when the window gets focus
onformchange	script	Triggers when a form changes
onforminput	script	Triggers when a form gets user input
onhaschange	script	Triggers when the document has change
oninput	script	Triggers when an element gets user input
oninvalid	script	Triggers when an element is invalid
onkeydown	script	Triggers when a key is pressed
onkeypress	script	Triggers when a key is pressed and released
onkeyup	script	Triggers when a key is released
onload	script	Triggers when the document loads
onloadeddata	script	Triggers when media data is loaded
onloadedmetadata	script	Triggers when the duration and other media data of a media element is loaded
onloadstart	script	Triggers when the browser starts to load the media data
onmessage	script	Triggers when the message is triggered
onmousedown	script	Triggers when a mouse button is pressed
onmousemove	script	Triggers when the mouse pointer moves
onmouseout	script	Triggers when the mouse pointer moves out of an element
onmouseover	script	Triggers when the mouse pointer moves over an element
onmouseup	script	Triggers when a mouse button is released
onmousewheel	script	Triggers when the mouse wheel is being rotated
onoffline	script	Triggers when the document goes offline
onoine	script	Triggers when the document comes online

ononline	script	Triggers when the document comes online
onpagehide	script	Triggers when the window is hidden
onpageshow	script	Triggers when the window becomes visible
onpause	script	Triggers when media data is paused
onplay	script	Triggers when media data is going to start playing
onplaying	script	Triggers when media data has start playing
onpopstate	script	Triggers when the window's history changes
onprogress	script	Triggers when the browser is fetching the media data
onratechange	script	Triggers when the media data's playing rate has changed
onreadystatechange	script	Triggers when the ready-state changes
onredo	script	Triggers when the document performs a redo
onresize	script	Triggers when the window is resized
onscroll	script	Triggers when an element's scrollbar is being scrolled
onseeked	script	Triggers when a media element's seeking attribute is no longer true, and the seeking has ended
onseeking	script	Triggers when a media element's seeking attribute is true, and the seeking has begun
onselect	script	Triggers when an element is selected
onstalled	script	Triggers when there is an error in fetching media data
onstorage	script	Triggers when a document loads
onsubmit	script	Triggers when a form is submitted
onsuspend	script	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate	script	Triggers when media changes its playing position
onundo	script	Triggers when a document performs an undo

onunload	script	Triggers when the user leaves the document
onvolumechange	script	Triggers when media changes the volume, also when volume is set to "mute"
onwaiting	script	Triggers when media has stopped playing, but is expected to resume

9. DHTML with JavaScript javaScript can create dynamic HTML content.
In JavaScript, the statement: document.write(), is used to write output to a web page
Example

```
<html>
<body>
<script type="text/javascript"> document.write(Date());
</script>
</body> </html>
```

A JavaScript can also be used to change the content or attributes of HTML elements. To change the content of an HTML element:

```
document.getElementById(id).innerHTML=new HTML
```

To change the attribute of an HTML element:

```
document.getElementById(id).attribute=new value
```

Visibility Example:

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<title>An Example for testing | QcTutorials</title>
</head>
<body>
<p id="p1">A text. A text. A text. A text. A text. A text. A text.</p>
<input type="button" value="Hide text"
onclick="document.getElementById('p1').style.visibility='hidden'" />
<input type="button" value="Show text"
onclick="document.getElementById('p1').style.visibility='visible'" />
</body>
</html>
```

Som of the events are
Events onload
onunload onchange

onsubmit onreset
onselect onblur
onfocus onkeydown
onkeyup onkeydown
vs onkeyup onkeypress
onmouseover &
onmouseout ondblclick
onmousedown & onmouseup
Disable right-click **10.**

Servlets:

What are Servlets?

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

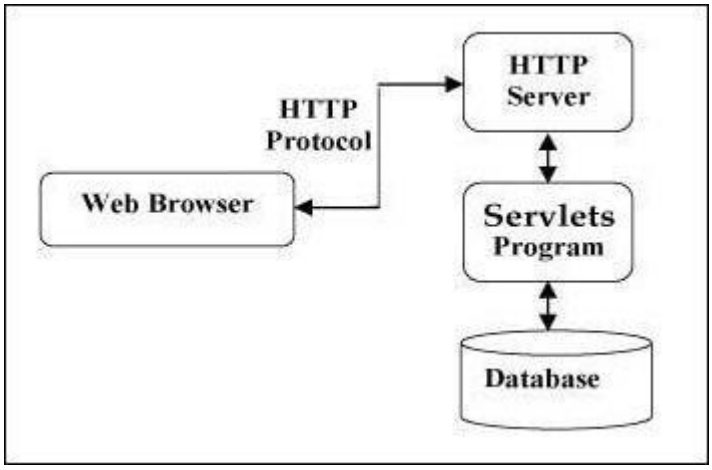
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.



Servlets Tasks

Servlets perform the following major tasks −

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.

- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

11. Servlet 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 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. web server) 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 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.

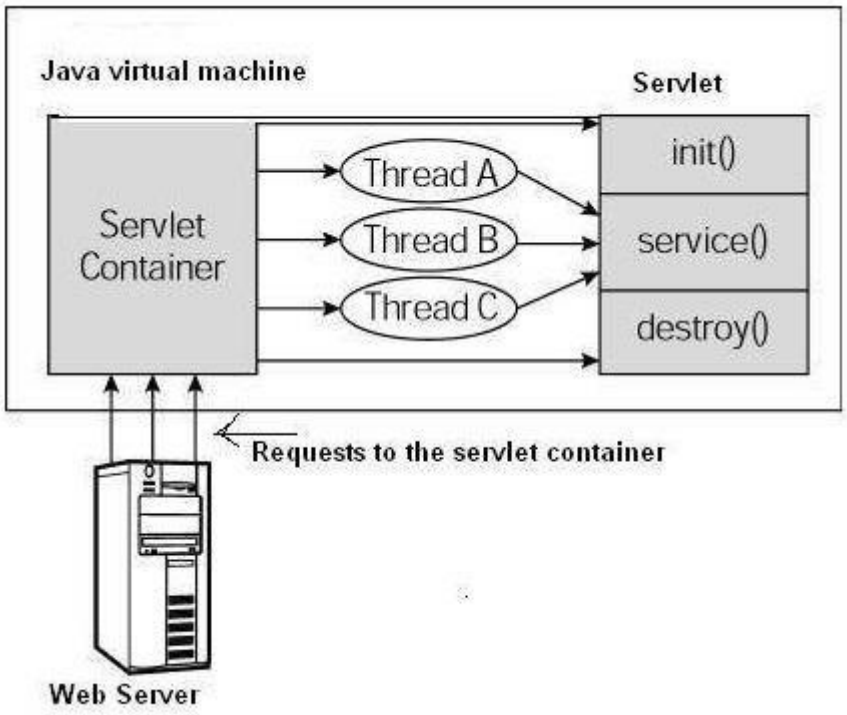
After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this −

```
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.



12. Form GET and POST actions

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 **?**(question mark) symbol as follows −

http://www.test.com/hello?key1 = value1&key2 = value2

OST 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 ? (question mark) in the URL it sends it as a separate message. 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-installationdirectory>/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**: samba
- **Last Name**: siva

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 imput.

```
<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.

13.Session Handling

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Still there are following three ways to maintain session between web client and 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 recieved cookie.

This may not be an effective way because many time browser does not support a cookie, so I would not recommend 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 POST data. Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.

This could 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 on the end of each URL that identifies the session, and the server can associate that session identifier with 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 it works even when browsers don't support cookies. The drawback of URL re-writing is that you would have to generate every URL dynamically to assign a session ID, even in case of a simple static HTML page.

Session Tracking Example

This example describes how to use the HttpSession object to find out the creation time and the lastaccessed time for a session.

```
// Import required java
libraries import java.io.*;
import javax.servlet.*; import
javax.servlet.http.*; import
java.util.*;

// Extend HttpServlet class
public class SessionTrack extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
```

```
    // Create a session object if it is already not  created.
    HttpSession session = request.getSession(true);

    // Get session creation time.
    Date createTime = new Date(session.getCreationTime());

    // Get last access time of this web page.
    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 web page.
    if (session.isNew()) {
      title = "Welcome to my website";
      session.setAttribute(userIDKey, userID);
    } else {
      visitCount = (Integer)session.getAttribute(visitCountKey);
visitCount = visitCount + 1;
      userID = (String)session.getAttribute(userIDKey);
    }
    session.setAttribute(visitCountKey,  visitCount);

    //     Set     response     content     type
response.setContentType("text/html");
PrintWriter out = response.getWriter();

    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" +
          "<h2 align = \"center\">Session Infomation</h2>\n" +
          "<table border = \"1\" align = \"center\">\n" +
```

```
                "<tr bgcolor = \"#949494\">\n" +
                  "          <th>Session      info</th><th>value</th>
</tr>\n" +
```

```
            "<tr>\n" +
              "  <td>id</td>\n" +
              "  <td>" + session.getId() + "</td>
            </tr>\n" +

            "<tr>\n" +
              "  <td>Creation Time</td>\n" +
              "  <td>" + createTime + "  </td>
            </tr>\n" +

            "<tr>\n" +
              "  <td>Time of Last Access</td>\n" +
              "  <td>" + lastAccessTime + "  </td>
            </tr>\n" +

            "<tr>\n" +
              "  <td>User ID</td>\n" +
              "  <td>" + userID + "  </td>
            </tr>\n" +

            "<tr>\n" +
              "  <td>Number of visits</td>\n" +
              "  <td>" + visitCount + "</td>
            </tr>\n" +
          "</table>\n" +
        "</body>
      </html>"
    );
  }
}
```

Compile the above servlet **SessionTrack** and create appropriate entry in web.xml file. Now running *http://localhost:8080/SessionTrack* would display the following result when you would run for the first time −

Welcome to my website	
Session Infomation	
Session info	**value**

id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

14 . Understanding Cookies.

Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets 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 or reset 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).

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.

Servlet Cookies Methods

Following is the list of useful methods which you can use while manipulating cookies in servlet.

Sr.No.	Method & Description
1	**public void setDomain(String pattern)** This method sets the domain to which cookie applies, for example tutorialspoint.com.
2	**public String getDomain()** This method gets the domain to which cookie applies, for example tutorialspoint.com.
3	**public void setMaxAge(int expiry)** This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session.
4	**public int getMaxAge()** This method returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.

5	**public String getName()** This method returns the name of the cookie. The name cannot be changed after creation.
6	**public void setValue(String newValue)** This method sets the value associated with the cookie
7	**public String getValue()** This method gets the value associated with the cookie.
8	**public void setPath(String uri)** This method sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.
9	**public String getPath()** This method gets the path to which this cookie applies.
10	**public void setSecure(boolean flag)** This method sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e. SSL) connections.
11	**public void setComment(String purpose)** This method specifies a comment that describes a cookie's purpose. The comment is useful if the browser presents the cookie to the user.
12	**public String getComment()** This method returns the comment describing the purpose of this cookie, or null if the cookie has no comment.

Example

Let us modify our Form Example to set the cookies for first and last name.

```
// 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 {

    // 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 );

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Setting Cookies Example";
    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>"
    );
  }
}
```

Compile the above servlet **HelloForm** and create appropriate entry in web.xml file and finally try following HTML page to call servlet.

```
<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>
```

15. Installing and Configuring Apache Tomcat Web Server

Installing Tomcat 7

There are certain steps we must follow for configuring Apache Tomcat 7.

Step 1

Download and Install Tomcat

1. Go to http://tomcat.apache.org/download-70.cgi then go to the Binary Distribution/Core/ and download the "zip" package (for example "apache-tomcat-7.0.40.zip", about 8MB).
2. Now **unzip** the downloaded file into a directory of our choice. Don't unzip onto the dekstop (since its path is hard to locate). I suggest using "e:\myserver". Tomcat will be unzipped into the directory "e:\myserver\tomcat-7.0.40".

Step 2

Check the installed directory to ensure it contains the following sub-directories:

bin folder

- logs folder
- webapps folder
- work folder
- temp folder
- conf folder
- lib folder

Step 3

Now, we need to create an Environment Variable JAVA_HOME.

We need to create an environment variable called "JAVA_HOME" and set it to our JDK installed directory.

1. To create the JAVA_HOME environment variable in Windows XP/Vista/7 we need to push the "Start" button then select "Control Panel" / "System" / "Advanced system settings". Then switch to the "Advanced" tab and select "Environment Variables" / "System Variables" then select "New" (or "Edit" for modification). In "Variable Name", enter "JAVA_HOME". In "Variable Value", enter your JDK installed directory (e.g., "c:\Program Files\Java\jdk1.7.0_{xx}").
2. For ensuring that it is set correctly, we need to start a command shell (to refresh the environment) and issue: set JAVA_HOME
   JAVA_HOME=c:\Program Files\Java\jdk1.7.0_{xx} <== Check that this is OUR JDK installed directory
3. Sometimes we need to set JRE_HOME also. So for creating JRE_HOME we need to use the same procedure. Push the "Start" buttonthen select "Control Panel" / "System" / "Advanced system settings". Then switch to the "Advanced" tab and select "Environment Variables" / "System Variables" then select "New" (or "Edit" for modification). In "Variable Name", enter

"JRE_HOME". In "Variable Value", enter your JRE installed directory (e.g., "C:\Program Files\Java\jre7\").

16. DATABASE CONNECTIVITY: JDBC perspectives

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for databaseindependent 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).

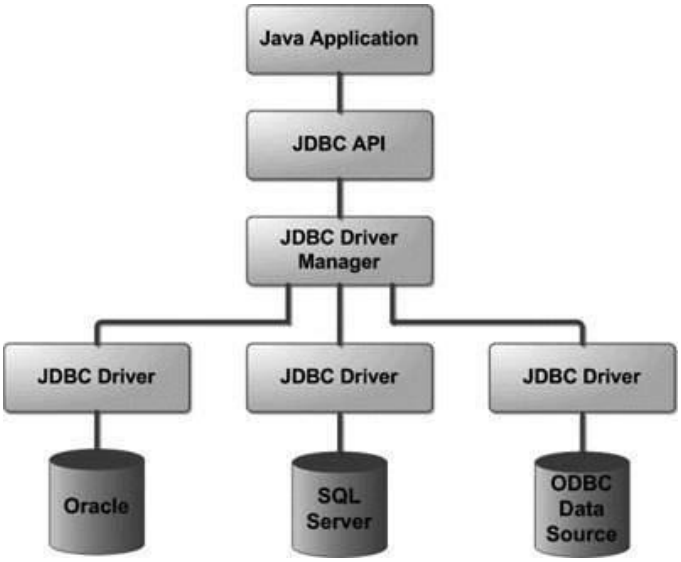
JDBC Architecture

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 −



17. JDBC program example an example on how to create a Database using JDBC application.

Before executing the following example, make sure you have the following in place −

- You should have admin privilege to create a database in the given schema. To execute the following example, you need to replace the *username* and *password* with your actual user name and password.
- Your MySQL or whatever database you are using, is up and running.

Required Steps

The following steps are required to create a new Database using JDBC application −

- **Import the packages:** Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.** will suffice.
- **Register the JDBC driver:** Requires that you initialize a driver so you can open a communications channel with the database.
- **Open a connection:** Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with the database server.

  To create a new database, you need not give any database name while preparing database URL as mentioned in the below example.
- **Execute a query:** Requires using an object of type Statement for building and submitting an SQL statement to the database.
- **Clean up the environment .** Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

Sample Code

Copy and past the following example in JDBCExample.java, compile and run as follows −

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost/";

  //  Database credentials     static final
String USER = "username";   static final
String PASS = "password";

  public static void main(String[] args) {
  Connection      conn      =      null;
Statement stmt = null;
  try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting  to  database...");          conn  =
DriverManager.getConnection(DB_URL, USER, PASS);

    //STEP 4: Execute a query
    System.out.println("Creating       database...");
stmt = conn.createStatement();

    String sql = "CREATE DATABASE STUDENTS";
stmt.executeUpdate(sql);
    System.out.println("Database created successfully...");
  }catch(SQLException se){
//Handle errors for JDBC
se.printStackTrace();
  }catch(Exception e){
    //Handle errors for Class.forName
e.printStackTrace();
```

```
   }finally{
      //finally block used to close
resources       try{          if(stmt!=null)
            stmt.close();
      }catch(SQLException
se2){       }// nothing we can do
try{          if(conn!=null)
            conn.close();
      }catch(SQLException se){
se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main
}//end JDBCExample
```

Now, let us compile the above example as follows −

```
C:\>javac JDBCExample.java

C:\>
```

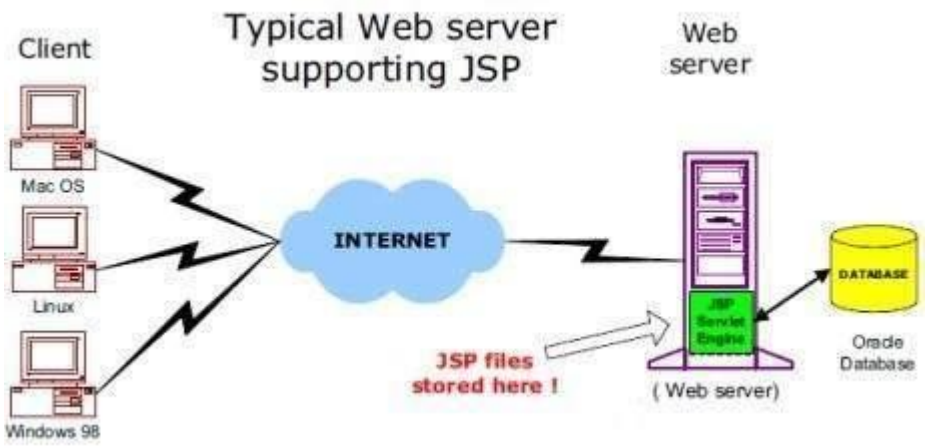
When you run **JDBCExample**, it produces the following result −

```
C:\>java JDBCExample Connecting to database...

Creating database...

Database created successfully...

Goodbye!

C:\>
```

20. JSP: Understanding Java Server Pages

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.
The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.



JSP Life cycle

A JSP life cycle is defined as the process from its creation till the destruction.

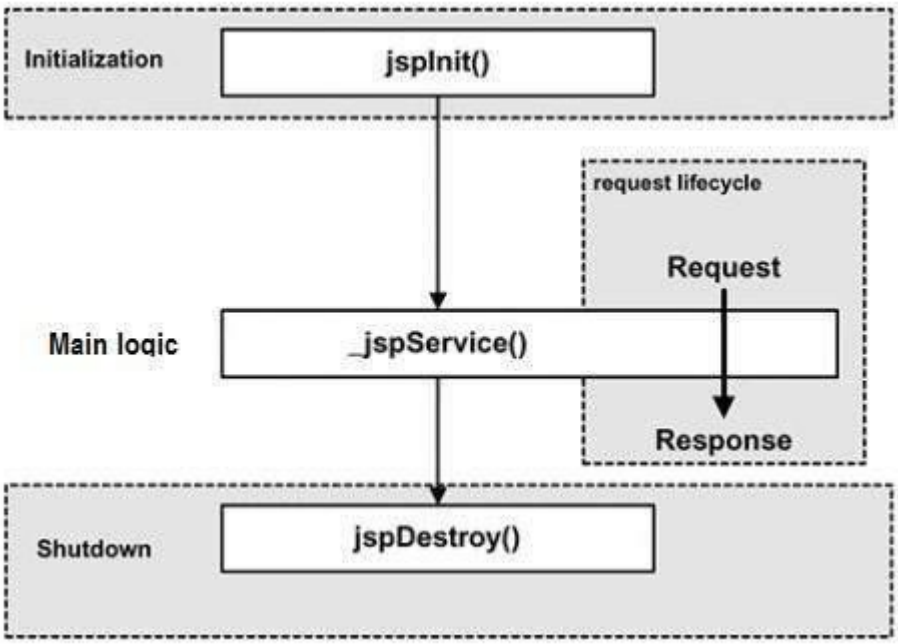
Paths Followed By JSP

The following are the paths followed by a JSP −

Compilation

- Initialization
- Execution
- Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below −



JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps −

Parsing the JSP.

- Turning the JSP into a servlet.
- Compiling the servlet. **JSP Initialization**

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method − public void jspInit(){
  // Initialization code...
}

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method. **JSP Execution**

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed. Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows − void _jspService(HttpServletRequest request, HttpServletResponse response) { // Service handling code... }

The **_jspService()** method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, **GET, POST, DELETE**, etc. **JSP Syntax**

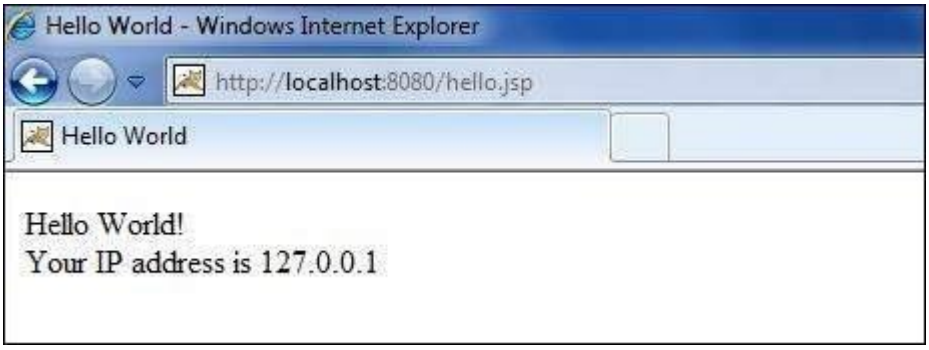
```
<% code fragment %>
```

first example for JSP −

```
<html>
  <head><title>Hello World</title></head>

  <body>
    Hello World!<br/>
    <%
      out.println("Your IP address is " + request.getRemoteAddr());      %>
  </body>
</html>
```

NOTE − Assuming that Apache Tomcat is installed in C:\apache-tomcat-7.0.2 and your environment is setup as per environment setup tutorial.

Let us keep the above code in JSP file **hello.jsp** and put this file in **C:\apachetomcat7.0.2\webapps\ROOT** directory. Browse through the same using URL **http://localhost:8080/hello.jsp**. The above code will generate the following result −



JSP Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax for JSP Declarations −

```
<%! declaration; [ declaration; ]+ ... %>
```

You can write the XML equivalent of the above syntax as follows −

```
<jsp:declaration>
code fragment
</jsp:declaration>
```

Following is an example for JSP Declarations −

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

S.No.	Syntax & Purpose
1	**<%-- comment --%>** A JSP comment. Ignored by the JSP engine.
2	**<!-- comment -->** An HTML comment. Ignored by the browser.
3	**<\%** Represents static <% literal.
4	**%\>** Represents static %> literal.

5	\' A single quote in an attribute that uses single quotes.
6	\" A double quote in an attribute that uses double quotes.

21 JSP Standard Tag Library(JSTL)

The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

Classification of The JSTL Tags

The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page − **Core Tags**

- **Formatting tags**
- **SQL tags**
- **XML tags**
- **JSTL Functions**

Core Tags

The core group of tags are the most commonly used JSTL tags. Following is the syntax to include the JSTL Core library in your JSP −

<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>

Following table lists out the core JSTL Tags −

S.No.	Tag & Description
1	**<c:out>** Like <%= ... >, but for expressions.
2	**<c:set >** Sets the result of an expression evaluation in a **'scope'**
3	**<c:remove >** Removes a **scoped variable** (from a particular scope, if specified).
4	**<c:catch>** Catches any **Throwable** that occurs in its body and optionally exposes it.
5	**<c:if>** Simple conditional tag which evalutes its body if the supplied condition is true.

6	**<c:choose>** Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by **<when>** and **<otherwise>**.
7	**<c:when>** Subtag of **<choose>** that includes its body if its condition evalutes to **'true'**.
8	**<c:otherwise >** Subtag of **<choose>** that follows the **<when>** tags and runs only if all of the prior conditions evaluated to **'false'**.
9	**<c:import>**
	Retrieves an absolute or relative URL and exposes its contents to either the page, a String in **'var'**, or a Reader in **'varReader'**.
10	**<c:forEach >** The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality .

Formatting Tags

The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Websites. Following is the syntax to include Formatting library in your JSP − <%@ taglib prefix = "fmt" uri = "http://java.sun.com/jsp/jstl/fmt" %>

Following table lists out the Formatting JSTL Tags −

S.No.	Tag & Description
1	**<fmt:formatNumber>** To render numerical value with specific precision or format.
2	**<fmt:parseNumber>** Parses the string representation of a number, currency, or percentage.
3	**<fmt:formatDate>** Formats a date and/or time using the supplied styles and pattern.
4	**<fmt:parseDate>** Parses the string representation of a date and/or time
5	**<fmt:bundle>** Loads a resource bundle to be used by its tag body.
6	**<fmt:setLocale>** Stores the given locale in the locale configuration variable.

SQL Tags

The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as **Oracle, mySQL**, or **Microsoft SQL Server**.

Following is the syntax to include JSTL SQL library in your JSP − <%@ taglib prefix = "sql" uri = "http://java.sun.com/jsp/jstl/sql" %>

Following table lists out the SQL JSTL Tags −

S.No.	Tag & Description
1	**<sql:setDataSource>** Creates a simple DataSource suitable only for prototyping
2	**<sql:query>** Executes the SQL query defined in its body or through the sql attribute.
3	**<sql:update>** Executes the SQL update defined in its body or through the sql attribute.
4	**<sql:param>** Sets a parameter in an SQL statement to the specified value.
5	**<sql:dateParam>** Sets a parameter in an SQL statement to the specified java.util.Date value.
6	**<sql:transaction >** Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

22. Creating HTML forms by embedding JSP code

Forms are an important aid to making the Web interactive. With JavaServer Pages, handling forms is easy--they do most of the work required to get to the information submitted with a form.

A Simple HTML Form

Main.jsp

```
<html>
  <head>
    <title>Using GET Method to Read Form Data</title>
  </head>
    <body>
    <h1>Using GET Method to Read Form Data</h1>
    <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>
```

GET Method Example Using Form

Following is an example that passes two values using the HTML FORM and the submit button. We are going to use the same JSP main.jsp to handle this input.

```
<html>
  <body>
        <form action = "main.jsp" 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>
```

UNIT III

Introduction to PHP: The problem with other Technologies (Servelets and JSP), Downloading, installing, configuring PHP, Programming in a Web environment and the anatomy of a PHP Page. Overview of PHP Data types and Concepts: Variables and data types, Operators, Expressions and Statements, Strings, Arrays and Functions. PHP Advanced Concepts: Using Cookies, Using HTTP Headers, Using Sessions, Authenticating users, Using Environment and Configuration variables, Working with Date and Time.

2.1 Server Side Programming

It is a technique used in Web design which involves embedded scripts in an HTML source code which results in a Client's request to the Server website being handled by a script/program running Server-Side before the Server responds to the client request.

Advantages of Server Side Programs:

- All programs reside in one machine called the Server. Any number of Clients can access the server programs.
- New functionalities to existing programs can be added at the server side.
- Migrating to new versions, architectures, design patterns, switching to new databases can be done at the Server side without having to bother about Clients.
- Issues relating to enter price applications like resource management, concurrency, session management, security and performance are managed by Server side applications.
- They are portable and possess the capability to generate dynamic and user-based content.

Some of the Server side programming Languages

ASP (Active Server Pages): This venerable Microsoft technology has been around since 1997, and was one of the first Web application technologies to integrate closely with theWeb server, resulting in fast performance. ASP scripts are usually written in VBScript, alanguage derived from BASIC.

This contrasts with PHP's more C - like syntax. Althoughboth languages have their fans, I personally find that it's easier to write structured,modular code in PHP than in VBScript.

ASP.NET: This is the latest incarnation of ASP, though in fact it's been rebuilt from theground up. It's actually a framework of libraries that you can use to build Web sites, andyou have a choice of languages to use, including C#, VB.NET (Visual Basic), and J#(Java). Because ASP.NET gives you a large library of code for doing things like creating**HTML forms** and accessing **database tables**, you can get a Web application up andrunning very quickly.

PHP, although it has a very rich standard library of functions, doesn't give you a structured framework to the extent that **ASP.NET** does.

Perl: Perl was one of the first languages used for creating dynamic Web pages, initially through the use of **CGI scripting** and, later, integrating tightly into Web servers with technologies like the Apache mod perl module and ActivePerl for IIS. Though Perl is a powerful scripting language, it's harder to learn than PHP. It's also more of a general – purpose language than PHP, although Perl's CPAN library includes some excellent modules for Web development.

Java: Like Perl, Java is another general - purpose language that is commonly used for Web application development. Thanks to technologies like JSP (*JavaServer Pages*) and servlets,Java is a great platform for building large - scale, robust Web applications. With software such as Apache Tomcat, you can easily build and deploy Java - based Web sites onvirtually any server platform, including Windows, Linux, and FreeBSD.

Disadvantages of Servlets:

- **Servlet** is a mixture of Java skills and web related HTML skills, because you haveto write the business logic in Java and the presentation you should use the HTML, sothe role based development is missing in pure Servlet. The developer who is writingServlet should know Java and HTML.
- If we used Servlet technology in our application, it is very difficult for **enhancement** and **bug fixing**.
- The servlet technology require more steps to develop, Servlet require too longtimefor development.
- Designing in Servlet is difficult and slow down the application.
- You need to a *Java Runtime environment*(JRE) on the server to run Servlet.

Disadvantages of JSP:

- JSP pages require about double the disk space to hold the page, Because JSP pagesare translated into class files, the server has to store the resultant class file with theJSP pages.
- JSP pages must be compiled on the server when first accessed this initial compilationproduces a noticeable delay when accessing the JSP page for the first time.
- JSP implementation typically issue poor diagnostics, Because JSP pages are translated, and then compiled into Java servlets, errors that creep in your pages are rarelyseen as errors arising from the coding of JSP pages.
- Lack of separation between presentation and logic that means providing multiplepresentations carries a very high cost.

Python: Conceived in the late 1980s, Python is another general - purpose programming language that is now commonly used to build dynamic Web sites. Although it doesn't havemuch in the way of Web - specific features built into the language, many useful modulesand frameworks, such as **Zope** and

Django, are available that make building Web applications relatively painless. Many popular sites such as Google and YouTube are built usingPython.

Python is a very nice language, but PHP is currently a lot more popular, and has a lotmore built - in functionality to help with building Web sites.

Ruby: Like Python, Ruby is another general - purpose language that has gained a lot oftraction with Web developers in recent years. This is largely due to the excellent *Ruby onRails'* application framework, which uses the Model - View - Controller (MVC) pattern,along with Ruby's extensive object - oriented programming features, to make it easy to build a complete Web application very quickly. As with Python, Ruby is fast becoming a popular choice among Web developers, but for now, PHP is much more popular.

ColdFusion: Along with **ASP**, **Adobe** ColdFusion was one of the first Web application frameworks available, initially released back in 1995. ColdFusion's main selling points are that it's easy to learn, it lets you build Web applications very quickly, and it's really easy to create database - driven sites.

An additional plus point is its tight integration with Flex, another Adobe technologythat allows you to build complex Flash - based Web applications. ColdFusion's maindisadvantages compared to PHP include the fact that it's not as popular (so it's harder to find hosting and developers), it's not as flexible as PHP for certain tasks, and the serversoftware to run your apps can be expensive. **Note:** ASP *and* ASP.NET *have a couple of other disadvantages compared to* PHP. *First of all, theyhave a commercial license, which can mean spending additional money on server software, and hostingis often more expensive as a result. Secondly,* ASP *and* ASP.NET *are fairly heavily tied to the Windowsplatform, whereas the other technologies in this list are much more cross - platform.*

2.2 Introduction to PHP

PHP stands for PHP: ***Hypertext Preprocessor***, which gives you a good idea of its corepurpose: to process information and produce hypertext (HTML) as a result.

- PHP is a *server - side scripting language* , which means that PHP scripts, or programs,usually run on a Web server.
- PHP is an *interpreted language* a PHP script is processed by the PHP engine each timeit ' s run.

PHP is a programming language for building dynamic, interactive Web sites. As a general rule,PHP programs run on a Web server, and serve Web pages to visitors on request. One of thekey features of PHP is that you can embed PHP code within HTML Web pages, making it veryeasy for you to create dynamic content quickly.

Although PHP only started gaining popularity with Web developers around 1998, it wascreated by **RasmusLerdorf** way back in 1994. PHP started out as a set of simple tools codedin the C language to replace the Perl scripts that **Rasmus** was using on his personal homepage (hence the original meaning of the ―PHP ― acronym). He released PHP to the generalpublic in 1995, and called it PHP version 2.

In 1997, two more developers, **ZeevSuraski** and **AndiGutmans**, rewrote most of PHPand, along with **Rasmus**, released PHP version 3.0 in June 1998. By the end of that year,PHP had already amassed tens of thousands of developers, and was being used on hundreds ofthousands of Web sites.

For the next version of PHP, **Zeev** and **Andi** set about rewriting the PHP core yet again, calling it the ―Zend Engine‖ (basing the name ―Zend‖ on their two names). The new version, PHP4, was launched in May 2000. This version further improved on PHP 3, and included sessionhandling features, output buffering, a richer core language, ISAPI support, Perl CompatibleRegular Expressions (PCRE) library and support for a wider variety of Web server platforms.

Although **PHP 4** was a marked improvement over version 3, it still suffered from a relativelypoor object - oriented programming (OOP) implementation. PHP 5, released in July 2004,addressed this issue, with private and protected class members; final, private, protected, andstatic methods; abstract classes; interfaces; and standardized constructor/destructor syntax.

PHP 5 was yet another watershed in the evolution of the PHP language. Although previous major releases had enormous numbers of new library additions, version 5 contained improvements over existing functionality and added several features commonly associated with mature programming language architectures like Vastly improved object-oriented capabilities, Try/catch exception handling, Improved XML and Web Services support and Native support for SQLite.

PHP 5.3 is actually the most significant upgrade to the language since the release of 5.0. Heralding a powerful array of new features including namespaces, late static binding, lambda functions and closures, a new MySQL driver, and a variety of syntactical additions such as NOWDOC syntax, version 5.3 represents a serious step forward in PHP's evolution.

A new major version of PHP known as **PHP 6** has been concurrently developed alongside PHP 5.X for several years, with the primary goal of adding Unicode support to the language. Although PHP 6 beta releases had previously been made available at http://snaps.php.net **2.2.1 Difference between HTML & PHP:**

Sno	HTML	PHP
1.	Hyper Text Markup Language	Hypertext Preprocessor (Personal Home Page)
2.	It is a **Markup** Language	It is a **Scripting** Language
3.	It is used to design a **Static Web pages**	It is used to design a **Dynamic Web pages**
4.	This code is executed in Client Machine (i.e Web Browser)	This code is executed in Server machine
5.	No additional software is used to execute.	Additionally PHP and Any server is needed to execute the code
6.	It does't converted to any language	After execution of php code is converted to html code.
7.	HTML is very easy and forgiving of mis-takes	Compare to html php is complex.
8.	HTML easier than php	learning time of PHP is longer than HTML
9.	HTML where anything you put in creates an output	PHP would not give you an output if something is wrong with your code
10.	Extension **.html**	Extension **.php**

Table 2.1: Difference between HTML and PHP **2.2.2**

Features of PHP:

a) Practicality

From the very start, the PHP language was created with practicality in mind. PHP's early evolution was not the result of the explicit intention to improve the language itself, but rather to increase its utility to the user. The result is a language that allows the user to build powerful applications even with a minimum of knowledge. For instance, a useful PHP script can consist of as little as one line; unlike C, there is no need for the mandatory inclusion of libraries.

PHP is a loosely typed language, meaning there is no need to explicitly create, typecast, or destroy a variable, although you are not prevented from doing so. PHP handles such matters internally, creating

variables on them as they are called in a script, and employing a best-guess formula for automatically typecasting variables. **b) Power**

PHP's ability to interface with databases, manipulate form information, and create pages dynamically, you might not know that PHP can also do the following:

- Create and manipulate Adobe Flash and Portable Document Format (PDF) files.
- Evaluate a password for guessability by comparing it to language dictionaries and easily broken patterns.
- Parse even the most complex of strings using the POSIX and Perl-based regular expression libraries.
- Authenticate users against login credentials stored in flat files, databases, and even Microsoft‘s Active Directory.
- Communicate with a wide variety of protocols, including LDAP, IMAP, POP3, NNTP, and DNS, among others.
- Tightly integrate with a wide array of credit-card processing solutions.

c) Possibility

- PHP developers are rarely bound to any single implementation solution. On the contrary, a user is typically fraught with choices offered by the language.
- PHP's flexible string-parsing capabilities offer users of differing skill sets the opportunity to not only immediately begin performing complex string operations but also to quickly port programs of similar functionality over to PHP.
- PHP offers comprehensive support for both procedural programming and object-oriented paradigm.
- The recurring theme here is that PHP allows you to quickly capitalize on your current skill set with very little time investment.

d) Price

PHP is available free of charge! Since its inception, PHP has been without usage, modification, and redistribution restrictions. In recent years, software meeting such open licensing qualifications has been referred to as open source software.

- **Free of licensing restrictions imposed by most commercial products:** Open source software users are freed of the vast majority of licensing restrictions one would expect of commercial counterparts.
- **Open development and auditing process:** Although not without incidents, open source software has long enjoyed a stellar security record. Such high-quality standards are a result of the open development and auditing process.
- **Participation is encouraged:** Development teams are not limited to a particular organization. Anyone who has the interest and the ability is free to join the project. The absence of member restrictions greatly enhances the talent pool for a given project, ultimately contributing to a higher-quality product.

2.3 The anatomy of a PHP Page

PHP documents end with the extension .php. When a web server encounters this extension in a requested file, it automatically passes it to the PHP processor. It need some syntactical rulesfor parsing PHP code.

2.3.1 Embedding PHP Code in Your Web Pages:

Default Syntax

The default delimiter syntax starts with <?php and ends with ? >, like this

<h3>Welcome!</h3>

<?php echo " Welcome to PHP

! ";

?>

* Some static output here! * ::::::::::

OUTPUT ::::::::::

Welcome to PHP !

Some static output here!

—<?php‖ This is due to the fact that PHP can be embedded within HTML Web pages. The final line of your simple script tells the PHP engine that it's reached the end of the current section of PHP code, and that the following lines (if any) contain plain HTML again: —? >‖

Short Tags

For less motivated typists, even shorter delimiter syntax is available. Known as short-tags, this syntax forgoes the php reference required in the default syntax, you may also encounter code where the opening and closing syntax used is like this:

<? echo "Welcome to

PHP !";

?>

:::::::::: **OUTPUT** ::::::::::

Welcome to PHP !

When short-tags syntax is enabled and you want to quickly escape to and from PHP to outputa bit of dynamic text, you can omit these statements using an output variation known as short-circuit syntax:

<?="This is another PHP example.";?>

Script

Certain editors have historically had problems dealing with PHP's more commonly used escapesyntax variants. Therefore, support for another mainstream delimiter variant, < script >, isoffered:

<script language="php">

print "This is another PHP example.";

</script>

:::::::::: **OUTPUT** ::::::::::

This is another PHP example.

ASP Style

Microsoft ASP pages employ a delimiting strategy similar to that used by PHP, delimitingstatic from dynamic syntax by using a predefined character pattern: opening dynamic syntaxwith < %, and concluding with %>. If youre coming from an ASP background and prefer tocontinue using this escape syntax, PHP supports it. Here's an example:

<%

print "This is another PHP example.";

%>

:::::::::: **OUTPUT** ::::::::::

This is another PHP example.

Keep in mind that just because you can do something doesn't mean you should. The ASP Styleand Script delimiting variants are rarely used and should be avoided unless you have amplereason for doing so.

Embedding Multiple Code blocks

You can escape to and from PHP as many times as required within a given page. For instance,the following example is perfectly acceptable:

```
<html>
<head>
<title><?php echo "Welcome to PHP!";?></title>
</head>
<body>
<?php
$date = "Jan 1, 2014";
?>
<p>Today's date is <?=$date;?></p>
</body>
</html>
```

:::::::::: **OUTPUT** ::::::::::

Today's date is Jan 1, 2014

2.3.2 Commenting Your Code

Whether for your own benefit or for that of somebody tasked with maintaining your code,the importance of thoroughly commenting your code cannot be overstated. PHP offers severalsyntactical variations for documenting your code.

Single-Line C++ Syntax

Comments often require no more than a single line. Because of its brevity, there is no need todelimit the comment's conclusion because the newline (\n) character fills this need quite nicely.PHP supports C++ single-line comment syntax, which is prefaced with a double slash (), likethis:

```
<?php
// Filename: firstpage.php
// This program printing a simple message echo
"This is a PHP program.";
?>
```

Shell Syntax

PHP also supports an alternative to the C++ -style single-line syntax, known as shell syntax,which is prefaced with a hash mark (#). Revisiting the previous example, Ill use hash marksto add some information about the script:

```
<?php
# Filename: firstpage.php
# This program printing a simple message echo
"This is a PHP program.";
?>
```

Multi-Line C Syntax

PHP also offers a multiple-line variant that can open and close the comment on different lines.Here is an example:

```
<?php
/*
Filename: firstpage.php
This program printing a simple message
*/
echo "This is a PHP program.";
?>
```

Output of above three programs is same i.e*This is a PHP program.*

2.3.3 Outputting Data to the Browser

print() statement

The print() statement outputs data passed to it . Its prototype looks like this:

int print(argument)

All of the following are possible print() statements:

```
<?php
print("<p>PHP was created by RasmusLerdorf.</p>");
?>
<?php
$str = "RasmusLerdorf";
print "<p>PHP was created by $str.</p>";
?> <?php
print "<p>PHP was created by RasmusLerdorf.</p>";
?>
```

All these statements produce identical output:*PHP was created by RasmusLerdorf.*

The print() statement's return value is misleading because it will always return 1 regardlessof outcome. **echo() statement**

Alternatively, you could use the echo() statement for the same purposes as print(). While thereare technical differences between echo() and print(), they'll be irrelevant to most readers andtherefore aren't discussed here. echo()'s prototype looks like this: *void echo(string argument1 [, ...string argumentN])*

To use echo(), just provide it with an argument just as was done with print():

```
echo "PHP was created by RasmusLerdorf.";
```

As you can see from the prototype, echo() is capable of outputting multiple strings. The utilityof this particular trait is questionable; using it seems to be a matter of preference more thananything else. Nonetheless, it's available should you feel the need. Here's an example:

```
<?php
$heavyweight = "Lennox Lewis"; $lightweight
= "Floyd Mayweather";
echo $heavyweight, " and ", $lightweight, " are great fighters.";
?>
```

This code produces the following: Lennox Lewis and Floyd Mayweather are great fighters.

The key difference between **echo()** and **print()** are

1. print() returns integer value echo() returns Boolean (i.e TRUE or FALSE) value.
2. Through print() we pass only one string, in echo() we pass multiple strings. **printf() statement**

The printf() statement is ideal when you want to output a blend of static text and dynamicinformation stored within one or several variables. It's ideal for two reasons. First, it neatlyseparates the static and dynamic data into two distinct sections, allowing for easy maintenance.Second, printf() allows you to have considerable control over how the dynamic information isrendered to the screen in terms of its type, precision, alignment, and position. Its prototypelooks like this: *integerprintf(string format [, mixed args])*

For example, suppose you wanted to insert a single dynamic integer value into an otherwisestatic string:

printf("one $ equals to %d rupees ", 62);

Executing this command produces the following:*one $ equals to 62 rupees*

In this example, **%d** is a placeholder known as a type specifier, andthe ***d*** indicates an integer value will be placed in that position. When the printf() statementexecutes, the lone argument, 100, will be inserted into the placeholder.

The key difference between **print()** and **printf()** are

1. In print() paranthesis are optional where as in printf() compulsary.
2. Through print() we pass only one string, in printf() we pass multiple strings.
3. print() gives unformatted output where as printf() gives formatted output through typespecifiers. The control strings are as follows

Type	**Description**
%b	Argument considered an integer; presented as a binary number
%d	Argument considered an integer
%c	Argument considered an integer; presented as a signed decimal number
%f	Argument considered a floating-point number; presented as a floating-point number
%o	Argument considered an integer; presented as an octal number
%s	Argument considered a string; presented as a string
%u	Argument considered an integer; presented as an unsigned decimal number
%x	Argument considered an integer; presented as a lowercase hexadecimal number
%X	Argument considered an integer; presented as an uppercase hexadecimal number

sprintf() statement

The sprintf() statement is functionally identical to printf() except that the output is assigned to a string rather than rendered to the browser. The prototype follows: *stringsprintf(string format [, mixed arguments]);* An example follows:

$cost = sprintf("$%.2f", 43.2); // $cost = $43.20

In this example, **%f** is a placeholder known as a type specifier, and the f indicates a float value will be placed in that position.

sprintf() print output to a variable, remaining three functions (i.e print(), echo(), and printf()) print output on Web browser.

3.1 OVERVIEW OF PHP DATA TYPES AND CONCEPTS

3.1.1. PHP variables [Compare with other language variables]:

The variables in PHP differ from the variables in other programming language such as C, C++ and Java. In all other languages variables are first assigned with data types and then initialized with a value.

In PHP, a data type of a variable is automatically assigned with a value making variables declaration and input of values easier. A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). PHP variable names are case-sensitive. Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

Example:

```
<?php
$x = 5; $y =
4; echo $x +
$y; $txt =
"Learning is
Interesting.‖;
echo "Learn
WT, $txt!";
?>
```

Output:

```
9
Learn WT, Learning is Interesting.
```

PHP is a Loosely Typed Language:

In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value. In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

Super global variables in PHP 5:

PHP 4.1.0 focus the concepts of Super global variables. Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special. The PHP super global variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST

- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

Example:
```
<?php
setcookie("user1","123");
setcookie("pass1","123");
$userid  = $_POST['uname'];
$passwd = $_POST['upwd'];
if ($_COOKIE['user1']==$userid  &&  $_COOKIE['pass1']==$passwd)
echo "User Authenticated. Success! ";  else
                    echo "User NOT Authenticated. Try again! ";
?>
```

3.1.2 PHP‟s Supported Datatypes:

A datatype is the generic name assigned to any data sharing a common set of characteristics.
Common datatypes include Boolean, integer, float, string, and array.
PHP supports eight primitive data types with three classifications.
Scalar Data types: Boolean, integer, float, and string
Compound data types: array and object
Special data types: resource and NULL **3.1.2.1**
Scalar Data type:

Scalar data types are used to represent a single value. Several data types fall under this category, including Boolean, integer, float, and string. PHP supports Four Scalar Data types such as Boolean, integer, float and string.

(i) Boolean:

The Boolean data type represents truth, supporting only two values: TRUE and FALSE (case insensitive). Alternatively, you can use zero to represent FALSE, and any nonzero value to represent TRUE. A few examples follow:

```
$var = false;  // $var is false.
$var = 1;        // $var is true. $var
= -1;   // $var is true. $var = 5;
        // $var is true.
$var = 0;          // $var is false.
```

(ii) Integer:

An integer is representative of any whole number or, in other words, a number that does not contain fractional parts. PHP supports integer values represented in base 10 (decimal), base 8 (octal), and base 16 (hexadecimal) numbering systems. Several examples follow:

```
42                  // decimal
```

-678900 // decimal
0755 // octal
0xC4E // hexadecimal

The maximum supported integer size is platform-dependent, although this is typically positive or negative 2^{31} for PHP version 5 and earlier. PHP 6 introduced a 64-bit integer value, meaning PHP will support integer values up to positive or negative 2^{63} in size.

(iii) Float:

Floating-point numbers, also referred to as floats, doubles, or real numbers, allow you to specify numbers that contain fractional parts. Floats are used to represent monetary values, weights, distances, and a whole host of other representations in which a simple integer value won‘t suffice.

PHP‘s floats can be specified in a variety of ways, several of which are demonstrated here:

Examples: 4.5678
4.0
8.7e4
1.23E+11

(iv) String:

A string is a sequence of characters treated as a contiguous group. Strings are delimited by single or double quotes. The following are all examples of valid strings:

Examples: "PHP is a great language"
'*9subway\n'
"123$%^789"

PHP treats strings in the same fashion as arrays. Example:

$color = "maroon";
$parser = $color[2]; // Assigns 'r' to $parser **3.1.2.2**

Compound Datatypes:

Compound datatypes allow for multiple items of the same type to be aggregated under a single representative entity. The array and the object fall into this category **(i) Array**

An array is an indexed collection of data values. Each member of the array index (also known as the key) references a corresponding value.

Example: $state[0] = "Alabama";
$state[1] = "Alaska"; $state[2]
= "Arizona";
...
$state[49] = "Wyoming";

(ii) Object

The other compound datatype supported by PHP is the object. The object is a central concept of the object-oriented programming paradigm. Unlike the other datatypes contained in the PHP language, an object must be explicitly declared. This declaration of an object‘s characteristics and behavior takes place within something called a class. Example: class Appliance { private $_power;
function setPower($status) {
$this->_power = $status;
}
}
...

$blender = new Appliance;
$blender->setPower("on"); **3.1.2.3**

Special Datatypes:

Special datatypes include: resource and NULL. The ‗resource' is mainly used in database-driven applications.

3.1.3. PHP Coercions [type conversion]:

Coercion, type coercion or type conversion is the process of altering the data type of a variable into another data type. PHP uses implicit type conversion (Weak/Loose Typing) so it is not necessary to declare the type of a variable. One data type can be converted into another implicitly or explicitly. If type conversion is implicit, then, it is called, Coercion. What coercion type is expected /required, is usually determined from the context of an expression. Coercions take place between the integer and double types, Boolean and other scalar types, and numeric and string types.

If a numeric value appears in string context, then that value is converted to string. Similarly, if a string value appears in numeric context, then that value is converted to a numeric value. If the string does not begin with a sign or digit, then non-numeric characters that follow the number in the string are ignored and zero is used.

Explicit Type Conversion:

There are THREE different ways to perform explicit type conversions,

(i) Using 'C' language syntax:

An expression can be cast to a different type using the syntax of ‗C'. Here, the required type name is written within the parentheses preceding the expression. For example, assume the value of $amt is 550.50. Then the result of the following will be 550 after casting double type into integer type.

Syntax: (int)$num;

(ii) Using Conversion Functions:

The conversion functions such as intval, doubleval or strval can be used, to specify explicit type conversion.

Syntax: intval($num);

(iii) Using Set type Function:

The settype function, takes two arguments. One is a variable and the Second is a string that specifies the required type name. It converts the value of the variables to the newly specified type.

Syntax: settype($amt, ―integer‖);

3.1.4. Scope and Lifetime of a PHP Variable:

(i) Scope of a Variable:

A variable defined in a function is called a local variable. The scope of the local variable is local to the function in which they are defined. A local variable cannot be used outside the function, even though it has the same name as a variable used outside the function.

A variable defined outside the function is called a Global variable. This variable can be accessed anywhere in the program. To access a global variable in a function, it must be declared in a global declaration of a function using, ‗global' keyword. Otherwise, the variable treated as a local variable only.

Example:
```
<?php
$x = 5; $y = 10;
function myTest() {
        global $x, $y;
$y = $x + $y;
} myTest();
echo $y;
?>
```

Output: 15

(ii) Life Time of a Variable:

The lifetime of a local variable in a PHP function begins, when the variable is first used and ends when the function execution terminates. That is after the termination of function execution, the value of local variable is lost. In order to retain this value throughout different calls to the same function, the function must declare local variable as _static‘.

The life time of a _static‘ local variable in a function begins when the variables is first used in the function but ends when the script execution ends. In PHP, the life time of static local variable ends, when the browser finishes execution of the document containing the embedded PHP script. In PHP function, a Static Local Variable is specified, using the static keyword. These variables are initialized only once when the first time the declaration is reached but they can be accessed any number of times.

Example:
```
<?php
function myTest()
{    static $x =
0;   echo $x;
   $x++; }
myTest();
myTest();
myTest();
?>
```

Outputs: 0
1
2

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called. The variable is still local to the function.

3.1.5. <u>PHP predefined Mathematical functions:</u>

PHP has several predefined functions, for number values. Some of the most common functions are,

abs — Absolute value acos — Arc cosine
asin — Arc sine atan — Arc tangent
base_convert — Convert a number between arbitrary bases bindec — Binary to decimal ceil — Round fractions up decbin — Decimal to binary dechex — Decimal to

hexadecimal decoct — Decimal to octal deg2rad — Converts the number in degrees to the radian exp — Calculates the exponent of e

floor — Round fractions down fmod — floating point remainder (modulo) of the division

hexdec — Hexadecimal to decimal log10 — Base-10 logarithm

log — Natural logarithm max — Find highest value

min — Find lowest value octdec — Octal to decimal

pi — Get value of pi pow — Exponential expression

rand — Generate a random integer round — Rounds a float

sin — Sine sinh — Hyperbolic sine

sqrt — Square root tan — Tangent

tanh — Hyperbolic tangent

Example: ***Output:***

```
<?php
$num1 = abs(-10);          // 10
$num2 = min(5,7,10,3,2);          // 2
$num1 = max(5,7,10,3,2);          // 10
$num1 = floor(7.999);      // 7
$num1 = ceil(7.999);       // 8
$num1 = round(5.45);       // 5.5
?>
```

3.2 OPERATORS

3.2.1. PHP Operator Types:

An *operator* is a symbol that tells the compiler to perform specific mathematical or logical manipulations. PHP language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

(i) Arithmatic Operators: There are following arithmetic operators supported by PHP language,

Operator	***Description***	***Example** [let $a=10 and $b=20]*
+	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 denominator	$b / $a will give 2
%	Modulus Operator gives remainder	$b % $a will give 0
++	Increment Operator (++i and i++)	$i++ which is equal to i=i+1
--	Decrement Operator (--i and i--)	$i-- which is equal to i=i-1

(ii) Comparison Operators: There are following comparison operators supported by PHP language,

Operator	***Description***	***Example*** *[let a=10 and b=20]*
==	Checks, two operands are equal or not	($a == $a) is not true.
!=	Checks, two operands are equal or not.	($a != $a) is true.
===	Checks, two operands are equal and have same data type.	($a === $b) is not true.
!==	Checks, two operands are not equal and have not same data type.	($a !== $b) is true.
>	Checks, left operand is greater than right operand.	($a > $b) is not true.
<	Checks, left operand is less than right operand.	($a < $b) is true.
>=	Checks, left operand is greater than or equal to right operand.	($a >= $b) is not true.
<=	Checks, left operand is less than or equal to right operand.	($a <= $b) is true.
? :	Ternary. ($a == $b) ? 5 : 10	$a==$b is true, return 5. Otherwise 10

(iii) Logical Operators: There are following logical operators supported by PHP language,

Operator	***Description***	***Example***	***Conclude***
AND equal.	Called Logical AND operator.	$a AND $b	true, if both a and b are
OR	Called Logical OR Operator.	$a OR $b	true, if either a or b is equal.
NOT equal.	Called Logical NOT Operator.	!($a && $b)	false, if both a and b are
&&	Called Logical AND operator.	$a && $b	true, if both a and b are equal.
||	Called Logical OR Operator.	$a || $b	true, if either a or b is equal.
!	Called Logical NOT Operator.	!($a && $b)	false, if both a and b are equal.
XOR true.	Called Logical XOR Operator.	$a XOR $b	true, if either a or b is

(iv) String Operators: There are following string operators supported by PHP language,

Operator	***Description***	***Example***	***Conclude***
=	Concatenation operator.	$a = ―ab‖ . ―cd‖;	assigns the string ―abcd‖ to $a.
.=	Concatenation - assignment.	$a .= ―efgh‖;	cat with previous value. Return ―abcdefgh‖

(v) Bitwise Operators: There are following bitwise operators supported by PHP language,

Operator	***Description***	***Example***	***Conclude***
&	called AND.	$a & $b	AND‘s each bit in $a and $b together.

|	called OR.	$a | $b	OR's each bit in $a and $b together.
^	called XOR	$a ^ $b	Exclusive OR's each bit in $a and $b together.
~	NOT	~$a	Negates each bit in $a.
<<	shift LEFT	<< $a	value of $a is shifted left.
>>	shift RIGHT	<< $a	value of $a is shifted right.

(vi) Assignment Operators: There are following assignment operators supported by PHP language,

Operator	***Description***	***Conclude***
=	Simple assignment operator.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator.	C %= A is equivalent to C

3.2.2. PHP Operators Categories:

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

i) Unary prefix operators, which precede a single operand.

ii) Binary operators, which take two operands and perform a variety of arithmetic and logical operations.

iii) The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.

iv) Assignment operators, which assign a value to a variable.

3.2.3. 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.

Operator	Associativity	Purpose
new	NA	Object instantiation
( )	NA	Expression subgrouping
[ ]	Right	Index enclosure
! ~ ++ --	Right	Boolean NOT, bitwise NOT, increment, decrement
@	Right	Error suppression
/ * %	Left	Division, multiplication, modulus
+ - .	Left	Addition, subtraction, concatenation
<< >>	Left	Shift left, shift right (bitwise)
< <= > >=	NA	Less than, less than or equal to, greater than, greater than or equal to
== != === <>	NA	Is equal to, is not equal to, is identical to, is not equal to
& ^ |	Left	Bitwise AND, bitwise XOR, bitwise OR
&& ||	Left	Boolean AND, Boolean OR
?:	Right	Ternary operator
= += *= /= .= %=&= |= ^= <<= >>=	Right	Assignment operators
AND XOR OR	Left	Boolean AND, Boolean XOR, Boolean OR
,	Left	Expression separation; example: `$days = array(1=>"Monday", 2=>"Tuesday")`

Table: Operator Precedence, Associativity, and Purpose

3.3 EXPRESSIONS

An expression is a statement used to perform actions in a program. It consists of one or more operators with at least one operand. Following are some example for an expression.

i) $a = 10; // This expression assigns integer value 10 to the variable $a

ii) $a = ―123‖; // This expression assigns string value ―123‖ to the variable $a iii) $menu = ―PUSH‖; // This expression assigns ―PUSH‖ to the variable $menu iv) $c=$a+$b; // This expression assigns sum of the value of $a and $b to the variable $c

v) $i++; // This expression post-increment the variable $a by 1.

3.4 STATEMENTS

3.4.1. Conditional / Selection Statements in PHP:

Conditional / Selection statements are used to perform different actions based on different conditions. In PHP we have the following conditional statements:

- **if statement** - executes some code only if a specified condition is true
- **if...else statement** - executes some code if a condition is true and another code if the condition is false

- **if...elseif....else statement** - specifies a new condition to test, if the first condition is false
- **switch statement** - selects one of many blocks of code to be executed **(i) if statement:**

If statement is a conditional statement. It is use to execute a statement when the condition of valuable is true. The if statement is used to execute some code **only if a specified condition is true**. Below is basic script of if statements.

```
Syntax:          if (condition)
 {
//      code      to      be      executed      if      condition      is      true;
 }
```

Example:

```
<?php
$t = date("H");
if ($t < "20")
{    echo "Have a good day!";    }
?>
```

Output: This php script will output "Have a good day!" if the current time (HOUR) is less than 20. **(ii)**

if…else statement:

If...else statement are a conditional statement which is use to execute a statement when condition of valuable is true and another statement will execute if valuable is false. The **if....else statement to execute some code if a condition is true and another code if the condition is false**. Below is basic script of if...else statement.

```
Syntax:          if (condition)
{      // code to be executed if condition is true;    } else
{       // code to be executed if condition is false;    }
```

Example:

```
<?php
$t = date("H"); if
($t < "20")
{    echo "Have a good day!";    } else
{  echo "Have a good night!";    }
?>
```

Output: This php script will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise.

(iii) if...elseif....else statement:

If...elseif...else statement are a conditional statement which is use to select one of several condition and execute the selected statement of condition. The if....elseif...else statement to **specify a new condition to test, if the first condition is false.** Below is basic script of if...elseif...else statement.

```
Syntax:          if (condition)
{      // code to be executed if condition is true;    } elseif(condition)
{      // code to be executed elseif condition is true;      } else
{       // code to be executed if condition is false;    }
```

Example:

```
<?php
$t = date("H");
if ($t < "10") {
echo "Have a good morning!";  elseif ($t < "20")
    echo "Have a good day!";
else
    echo "Have a good night!";
?>
```

Output: This php script will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!". **(iv) switch…case statement:**

Switch statement are same with If...elseif...else statement, but switch statement only use to comparing single variables with some values is there. The switch statement is to **select one of many blocks of code to be executed**. Below is basic script of switch statement.

Syntax:

```
switch (n) {
  case label1: code to be executed if n=label1;
               break;
  case label2: code to be executed if n=label2;
               break;
  case label3: code to be executed if n=label3;
               break;
               ...
               …
  default:     code to be executed if n is different from all labels;
  }
```

Example:

```
<?php
$favcolor = "red";
switch ($favcolor)
{ case "red": echo "Your favorite color is red!";
              break;
  case "blue":    echo "Your favorite color is blue!";
              break;
  case "green":       echo "Your favorite color is green!";
              break;
  default:    echo "Your favorite color is neither red, blue, or green!";
              break;
}
?>
```

Output: Your favorite color is red!

3.4.2. <u>Loop Statements in PHP:</u>

There is some situation to execute same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this. In PHP, we have the following looping statements:

- while - loops through a block of code as long as the specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array
- Break & Continue Statements - used to terminate the execution of a loop prematurely & halt the current iteration of a loop but it does not terminate the loop.

(i) 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.

Prototype:
```
while (condition)
{
        //  code to be executed;
}
```
Example:
```
<?php
$i = 0;
$num = 50;
while( $i < 10)
{
        $num--;
        $i++;
}
echo ("Loop count = $i and num = $num" );
?>
```
Output: Loop count = 10 and num = 40

(ii) do...while loop statement:

The Do...While statements are similar to While statements, except that the condition is tested at the end of each iteration, rather than at the beginning. This means that the Do...While loop is guaranteed to run at least once. 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. The Do...While loop syntax is,

Prototype:
```
do
{
        //  code to be executed;
}while (condition);
```
Example:
```
<?php
$i = 0;
$num = 50;
do
{
```

```
        $num--;
        $i++;
    }while( $i > 10 );
    echo ("Loop count = $i and num=$num" );
    ?>
```

Output: Loop count = 0 and num=49

(iii) 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. For this reason, the For loop is known as a *definite loop.* The For loop syntax is as follows,

Prototype:
```
for (initialization; condition; increment)
{
    // code to be executed;
}
```

The For statement takes three expressions inside its parentheses, separated by semi-colons. When the For loop executes, the following occurs.

- The initializing expression is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity.
- The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the For loop terminates.
- The update expression increment executes.
- The statements execute, and control returns to step 2.

Example:
```
<?php
$num = 4;
$fact = 1;
for($i=0; $i<$num; $i++)
{
    $fact = $fact * $i;
}
echo ―The Factorial value of $num ! = $fact‖;
?>
```

Output: The Factorial value of 4! = 24

(iv) foreach loop statement:

The ―foreach‖ loop is a variation of the For loop and allows you to iterate over elements in an array. 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. There are two different versions of the ―foreach‖ loop. The ―foreach‖ loop syntaxes are as follows:

Prototype 1:
```
foreach (array as value)
{
    // code to be executed;
}
```

Prototype 2:
```
foreach (array as key => value)
{
    // code to be executed;
}
```

Example:

```
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
    echo "Value is $value <br />";
?>
```

Output:

Value is 1

Value is 2

Value is 3

Value is 4

Value is 5

Note: In ―foreach (array as key => value)‖ case, the key for each element is placed in $key and the corresponding value is placed in $value. The Foreach construct does not operate on the array itself, but rather on a copy of it. During each loop, the value of the variable $value can be manipulated but the original value of the array remains the same.

(v) Break & Continue statement:

The PHP **break** keyword is used to terminate the execution of a loop prematurely. The **break** statement is situated inside the statement block. If 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:

```
<?php
$i = 0;
while( $i < 10)
{
        $i++;
        if( $i == 3 )
            break;
}
echo ("Loop stopped prematurely at i = $i" );
?>
```

Output: Loop stopped prematurely at i = 3

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: $array = array(1, 2, 3, 4, 5); foreach ($array as $value)

```
{
        if( $value == 3 )
        {   continue; }
```

```
        echo "Value is $value <br />";
    }
    ?>
```

Output:

```
Value is 1
Value is 2
Value is 4
Value is 5
```

The „goto" Statements

The goto operator can be used to jump to another section in the program. The target point is specified by a label followed by a colon, and the instruction is given as goto followed by the desired target label. This is not a full unrestricted goto. The target label must be within the same file and context, meaning that you cannot jump out of a function or method, nor can you jump into one. You also cannot jump into any sort of loop or switch structure. You may jump out of these, and a common use is to use a goto in place of a multi-level break. An example follows:

```
<?php
for ($count = 0; $count < 10; $count++)
{
$randomNumber = rand(1,50);
if ($randomNumber < 10)
goto less; else
echo "Number greater than 10: $randomNumber<br />";
} less:
echo "Number less than 10: $randomNumber<br />";
?>
```

The include() Statement

The include()statement will evaluate and include a file into the location where it is called. Including a file produces the same result as copying the data from the file specified into the location in which the statement appears. Its prototype follows: *include(/path/to/filename);* Example: <?php

```
include "/usr/local/lib/php/wjgilmore/init.inc.php";
/* the script continues here */
?>
```

You can also execute include()statements conditionally.

Note: Any code found within an included file will inherit the variable scope of the location of its caller.

Ensuring a File Is Included Only Once

The include_once()function has the same purpose as include()except that it first verifies whether the file has already been included. Its prototype follows: *include_once (filename);*

If a file has already been included, include_once()will not execute. Otherwise, it will include the file as necessary. **Requiring a File**

For the most part, require()operates like include(), including a template into the file in which the require()call is located. Its prototype follows: *require (filename);*

However, there are two important differences between require()and include(). First, the file will be included in the script in which the require()construct appears, regardless of where require()is located.

The second important difference is that script execution will stop if a require() fails, whereas it may continue in the case of an include().

Ensuring a File Is Required Only Once

As your site grows, you may find yourself redundantly including certain files. Although this might not always be a problem, sometimes you will not want modified variables in the included file to be overwritten by a later inclusion of the same file. Another problem that arises is the clashing of function names should they exist in the inclusion file. You can solve these problems with the require_once()function. Its prototype follows: *require_once (filename);*

The require_once()function ensures that the inclusion file is included only once in your script. After require_once()is encountered, any subsequent attempts to include the same file will be ignored. **The „return" Statement**

If called from within a function, the return statement immediately ends execution of the current function, and returns its argument as the value of the function call. return will also end the execution of an eval() statement or script file.

If called from the global scope, then execution of the current script le is ended. If the current script file was included or required, then control is passed back to the calling file. Furthermore, if the current script file was included, then the value given to return will be returned as the value of the include call. If return is called from within the main script file, then script execution ends.

If the current script file was named by the auto prepend file or auto append file configuration options in php.ini, then that script file's execution is ended.

3.5 STRINGS

There are three ways to write a literal string in your program: using single quotes, double quotes, and the here document (heredoc) format derived from the Unix shell. These methods differ in whether they recognize special escape sequences that let you encode other characters or interpolate variables.

3.5.1 Single-Quoted Strings:

Single-quoted strings do not interpolate variables. Thus, the variable name in the following string is not expanded because the string literal in which it occurs is single quoted: $name = 'Fred';

$str = 'Hello, $name'; // single-quoted echo

$str;

OUTPUT: Hello, $name

The only escape sequences that work in single-quoted strings are — ' ‖, which puts a single quote in a single-quoted string, and \n, which puts a backslash in a single-quoted string. Any other occurrence of a backslash is interpreted simply as a backslash:

$name = 'Tim O\'Reilly'; // escaped single quote echo

$name;

$path = 'C:\\WINDOWS'; // escaped backslash echo

$path;

$nope = '\n'; // not an escape echo

$nope;

3.5.2 Double-Quoted Strings:

Strings enclosed in double quotes are the most commonly used in PHP scripts because they offer the most flexibility. Escape sequences are also parsed. Consider this example: <?php

$output = "This is one line.\n And this is another line."; echo
$output;
?>

3.5.3 Here Documents (heredocs):

You can easily put multiline strings into your program with a heredoc, as follows:

$clerihew = <<< Identifier
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.
identifier; echo
$clerihew; The <<< identifier token tells the PHP parser that you're writing a **heredoc**. There must be a space after the <<< and before the identifier. You get to pick the identifier. The next line starts the text being quoted by the **heredoc**, which continues until it reaches a line that consists of nothing but the identifier.

As a special case, you can put a semicolon after the terminating identifier to end the statement, as shown in the previous code. If you are using a **heredoc** in a more complex expression, you need to continue the expression on the next line, as shown here: printf(<<< Template %s is %d years old.
Template
, "Fred", 35);

3.5.4 String Manipulation Functions:

Length of a String

In PHP we use the function strlen() to determine the length of the String(s). I treturns number of characters in a string including spaces. The prototype is as follows:

int strlen(string str);

The following example to check length of user name

```
<?php
$uname = "dontap"; if
(strlen($uname) <= 10)
echo "uname is too short!";
else
echo "Uname is valid!";
?>
```

Comparing two strings

In PHP, Comparing two strings using strcmp() function. It compares two strings with case-sensitive. The prototype is as follows:

int strcmp(string str1, string str2)

It returns three possible values: 0 if str1 and str2 are same, -1 if str1 is less than str2 and 1 if str1 is greater than str2. The following example demonstrate comparing password and confirm password:

```
<?php
$pwd = "dontap"; $cpwd =
"donthap"; if (strcmp($pwd,
$cpwd) != 0) {
echo "Password and Confirm-password do not match!";
} else {
echo "Password and Confirm-password match!";
}
?>
```

Changing Case

PHP has several functions for changing the case of strings: strtolower() and strtoup per() operate on entire strings, ucfirst() operates only on the first character of the string, and ucwords() operates on the first character of each word in the string. Each function takes a string to operate on as an argument and returns a copy of that string, appropriately changed. For example:

```
$string1 = "FRED flintstone"; $string2
= "barney rubble";
print(strtolower($string1));
print(strtoupper($string1));
print(ucfirst($string2));
print(ucwords($string2));
```

Removing HTML tags from strings

The strip tags() function removes HTML tags from a string.

```
$input = '<p>Howdy, &quot;Cowboy&quot;</p>';
$output = strip_tags($input);
// $output is 'Howdy, &quot;Cowboy&quot;'
```

The function may take a second argument that specifies a string of tags to leave in the string. List only the opening forms of the tags. The closing forms of tags listed in the second parameter are also preserved:

```
$input = 'The <b>bold</b> tags will <i>stay</i><p>';
$output = strip_tags($input, '<b>');
// $output is 'The <b>bold</b> tags will stay'
```

Attributes in preserved tags are not changed by strip tags(). Because attributes such as style and onmouseover can affect the look and behavior of web pages, preserving some tags with strip tags() won't necessarily remove the potential for abuse.

3.6 ARRAYS

3.6.1 Array Basics:

An array is a single variable that can hold more than one value at once. You can think of an array as a list of values. Each value within an array is called an element, and each element is referenced by its own index , which is unique to that array. To access an element's value whether you're creating, reading, writing, or deleting the element you use that element's index.

Note: An array index is often referred to as a key. Typically, a numeric index is called an index and a string index is called a key; however there's no hard - and - fast rule with this.

In PHP, There are three kinds of Arrays:-

- **Numeric Array:** An Array with numeric Key (index).
- **Associative Array:** An array where each key is associated with a value.
- **Multi-dimensional Array:** An array containing one or more arrays.

Arrays in PHP are flexible and built-in data structure. They are entirely different from the arrays found in any other programming language. PHP‘s array can be defined as, a combination of arrays found in a typical language such as C and associative arrays or hashes available in other languages such as, Perl, Ruby and Python.

In PHP, an array element consists of two parts a _Key‘ and a _Value‘. Another interesting feature of arrays is that, keys can be only positive integers, only string or a combination of both i.e., some of its elements have integer keys and some have string keys **Creating Arrays:**

Individual elements of a PHP array are referenced by denoting the element between a pair of square brackets. Because there is no size limitation on the array, you can create the array simply by making reference to it, like this:

```
$branch[0] = "CSE";
```

Additional values can be added by mapping each new value to an array index, like this:

```
$branch[1] = "EEE"; $branch[2] = "ECE";
...
$branch[10] = "MECH";
```

Interestingly, if you intend for the index value to be numerical and ascending, you can omit the index value at creation time:

```
$branch[] = "CSE"; $branch[]
= "ECE";
...
$branch[] = "EEE";
```

Creating Arrays with array()

The simplest way to create a new array variable is to use PHP's built - in array() construct. This takes a list of values and creates an array containing those values, which you can then assign to a variable:

```
$branch = array( "CSE", "ECE","EEE","MECH" );
```

In this line of code, an array of four elements is created, with each element containing a string value. The array is then assigned to the variable **$branch**. You can now access any of the array elements via the single variable name, **$branch**, as you see in a moment.

If you want to create an associative array, where each element is identified by a string index rather than a number, you need to use the => operator, as follows:

$student = array("Name" = > "Suresh", "branch" = > "CSE", "Gender" => "Male");

This creates an array with three elements: ―Suresh‖, which has an index o f‖ Name‖; ―CSE‖, which has an index of ‖branch‖ ; and ―Male‖, which has an index of ―Gender‖.

Multidimensional arrays

In Multi-dimensional array elements in the array can be an array, and each element in the sub-array can be an array and so on.

Example: In this example we create multidimensional array, with automatically assigned keys.

```
<?php
$course=array("UG"=>array("CSE","ECE","EEE","MECH"),
              "PG"=>array("M.Tech(CSE)","M.Tech(CS)","M.Tech(VLSI)"),
"Dip"=>array("DCME","DECE","DEEE"));
?>
```

The above array would look like this it written to the output:

```
Arry([UG]=>Array( [0]=>CSE
[1]=>ECE
[2]=>EEE
[3]=>MECH)
                    [PG]=>Array( [0]=>M.Tech(CSE)
[1]=>M.Tech(CS)
[2]=>M.Tech(VLSI))
[Dip]=>Array([0]=>DCME
[1]=>DECE
[2]=>DEEE))
```

Accessing an Array Element:

Accessing of array elements is the same as in other languages. The brackets with the subscript of the key are used to access an individual array element. This is irrespective of the integer key or the string key.

```
$score[_sub1‘] = 55;
$day[0] = ―Sunday‖;
```

Outputting an Array:

The method that is used very often to output the contents of array is to iterate over each key and echo the corresponding value. This can be done using *“foreach”* statement.

Example:

```
<?php
$subjects = array(―WT‖, ―ST‖, ―DWDM‖, ―SPM‖, ―MEFA‖);  foreach($subjects as $subject)
                    {       echo ―{ $subject } <br />‖;    }
```

Printing Arrays for Testing purposes:

The print_r() function accepts a variable and sends its contents to standard output, returning TRUE on success and FALSE otherwise.

Prototype: Boolean print_r(mixed variables [, Boolean return]);

The print_r function takes the variables, sends the variable contents to standard output and return TRUE if the operation is successful else it returns FALSE. In addition to this, print_r() function builds the contents of array into readable format. Consider the example where the contents of associative array holding few subjects and their subject_ids are to be viewed.

Example:

```
<?php
$subjects = array(—WT‖, —ST‖, —DWDM‖, —SPM‖, —MEFA‖);
print_r($subjects);
?>
```

Output: Array([0] => WT, [1] => ST, [2] => DWDM, [3] => SPM, [4[=> MEFA)

3.6.2 Adding and Removing Array Elements:

PhP provides a number of functions for both growing and shrinking an array. Some of these functions are provides as a convenience to programmers to do various queue implementations. **i) Adding a value to the Front of an Array:**

The array_unshift() function adds elements to the front of an array. All preexisting numerical keys are modified to reflect their new position in the array, but associative keys aren‘t affected.

Prototype: *int array_unshift(array array, mixed variables[, mixed variables .....]);* **ii) Adding a value at the End of an Array:**

The array_push() function adds a value at the end of an array. Returning the total count of elements in the array after the new value has been added.

Prototype: *int array_push(array array, mixed variables[, mixed variables .....]);* **iii) Removing a value from the Front of an Array:**

The array_shift() function removes and returns the first item found in an array. If numerical keys are used, all corresponding values will be shifted down, whereas arrays using associative keys will not be affected.

Prototype: *mixed array_shift(array array);* **iv) Removing a value at the End of an Array:**

The array_pop() function removes and returns the last element in an array. Its prototype as follows, Its prototype as follows,

Prototype: *mixed array_pop(array array);*

Example:

```
<?php
$num = array(1,2,3,4,5);
array_unshift($num, -1, 0); foreach
($num as $nums)
{ echo "{$nums } "; } echo "<br />"; }
array_push($num, 6, 7);
foreach ($num as $nums)
{ echo "{$nums } "; } echo "<br />"; }
array_shift($num);
foreach ($num as $nums)
{ echo "{$nums } "; } echo "<br />"; }
array_pop($num);
foreach ($num as $nums)
{ echo "{$nums } "; } echo "<br />"; }
```

```
?>
```

Output:

```
-1 0 1 2 3 4 5
-1 0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6
```

3.6.3 Locating / Searching Array Elements:

PHP introduces several functions that enable to search arrays in order to locate array elements. Some of them are as follows, **i) Searching an Array:**

The function in_array() searches an array for a specific value, returning TRUE if the value is found and FALSE otherwise.

Prototype: *Boolean in_array(mixed needle, array haystack[, Boolean strict]);* **ii) Searching Associative Array Keys:**

The function array_key_exists() returns TRUE if a specified key is found in an array and FALSE otherwise.

Prototype: *Boolean array_key_exists(mixed key, array array);* **iii) Searching Associative Array Values:**

The function array_search() searches an array for a specified value, returning its key located and FALSE otherwise.

Prototype: mixed *array_search(mixed needle, array haystack [, boolean strict]);* **iv) Retrieving Array Keys:**

The function array_keys() returns an array consisting of all keys located in an array. Its prototype as follows,

Prototype: array *array_keys(array array [, mixed search_value [, boolean preserve_keys]]);*

v) Retrieving Array Values:

The function array_values() returns all values located in an array, automatically providing numeric indexes for the returned array. Its prototype as follows,

Prototype: array *array_values(array array);*

Example:

```
<?php
$num = array(1,2,3,4,5);
$key = 3;
if(in_array($key, $num))
    echo " Search element found!";
else
    echo " Search element not found!";
?>
```

Output: Search element found!

3.6.4 Traversing an Array Elements:

PHP introduces several functions that enable to Traverse an array in order to Retrieving the Key or Value, Current pointer location and Moving pointer location to next or previous pointer. Some of them are as follows,

i) Retrieving the Current Array Key:

The function key() returns the key located at the current pointer position of the provided array. Its prototype is as follows,

Prototype: mixed key(array array); **ii)**

Retrieving the Current Array Value:

The function current() returns the array value residing at the current pointer position of an array. Its prototype is as follows,

Prototype: mixed current(array array); **iii) Moving**

the pointer to the NEXT Array position:

The function next() returns the array value residing at the pointer position immediately following that of the current array pointer. Its prototype is as follows, Prototype: mixed next(array array); **iv) Moving the pointer to the PREVIOUS Array position:**

The function prev() returns the array value residing at the location preceding the current pointer location, or FALSE if the pointer resides at the first position in an array. Its prototype is as follows,

Prototype: mixed prev(array array);

v) Moving the pointer to the FIRST Array position:

The function reset() serves to set an array pointer back to the beginning of the array. Its prototype is as follows,

Prototype: mixed reset(array array); **vi) Moving the**

pointer to the END Array position:

The function end() moves the pointer to the last position of an array, returning the last element. Its prototype is as follows,

Prototype: mixed end(array array);

3.6.5 Sorting an Array Elements:

PHP introduces several functions that enable to Sort array and its elements. The elements in an array can be sorted in alphabetical or numerical order, descending or ascending. The following are some PHP array sort functions,

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

Example:

```
?php
$cars = array("Volvo", "BMW", "Toyota");
$numbers = array(4, 6, 2, 22, 11);
sort($cars);        $len        =
count($cars); for($x = 0; $x
< $len; $x++)          {
echo $cars[$x];
      echo "<br>";                }
print_r(rsort($numbers));
?>
```

Output:

```
Volvo
Toyota
BMW
```

Array([0] => 22, [1] => 11, [2] => 6, [3] => 4, [4[=> 2) **3.6.6**

Some more Array fuctions on array elements:

(i) Shuffle:

This function shuffles (randomizes the order of the elements in) an array. Syntax: *boolean **shuffle** (array &$array);*

Example:

```
<?php
$numbers = range(1, 10);
shuffle($numbers);
foreach ($numbers as $number)
{   echo "$number "; }
?>
```

Output: *2 5 7 9 10 1 3 4 6 8*

ii) Merger:

The array_merge() array function in PHP will merge multiple arrays. This function takes a list of arrays separated by commas as its parameters. Syntax: *array array_merge (array $array1 [, array $...])*;

Example:

```
<?php
$array1 = array("orange", "apple", "grape");

$array2 = array("peach", 88, "plumb");

$array3 = array("lemon", 342);

$newArray = array_merge($array1, $array2, $array3);

foreach ($newArray as $key => $value)

    echo "$key - $value <br />";

?>
```

Output: plumb 0 - orange 1 - apple 2 - grape 3 - peach 4 - 88 5 - 6 - lemon 7 – 342

iii) Combine:

The array_combine() function creates an array by using the elements from one "keys" array and one "values" array. Both arrays must have equal number of elements. Syntax:

array_combine(keys,values);

Example:

```
<?php
$fname=array("Peter","Ben","Joe");
$age=array("35","37","43");
$c=array_combine($fname,$age);
print_r($c);
?>
```

Output: Array ([Peter] => 35 [Ben] => 37 [Joe] => 43)

iv) Array Sum:

The array_sum() returns the sum of values in an array. Syntax: number array_sum (array $array);

Example:
```
<?php
 $a = array(2, 4, 6, 8);    echo "sum(a) = " . array_sum($a) .
"\n";   $b = array("a" => 1.2, "b" => 2.3, "c" => 3.4);
                echo "sum(b) = " . array_sum($b) . "\n";
                ?>
```
Output:
```
sum(a) = 20
sum(b) = 6.9
```

3.7 FUNCTIONS

Function is a set of statements that perform a specific task and can be executed at any time. PHP defines several built in functions and also supports user –defined function. The general syntax of a PHP function is as follows,

Prototype:
```
function function_name(args 1, args 2, …., args n )
{
        // block of codes / statements;
}
```

PHP function definitions, starts with the keyword function, which indicates, that the following is the function definition. A function name is the meaningful and unique name of the function, which always begins with a letter or an underscore followed by any number of letters or digits.

The function names are case-insensitive meaning that, both the functions add() and Add() are the same. If the script includes two functions with the same name, then the PHP interpreter reports an error message stating that there are two definitions for the same function. An error is also reported if there are two functions with the same name, even differ in their parameters list because PHP does not support function overloading.

A list of parameters enclosed within parentheses is optional because, it is not necessary for a function‘s definition to appear in a document before it is referenced. If a document uses (or) more related functions, then their definitions must be defined in separate files and they should be called in the document using the include function.

A function may also return a value to the caller. The return statement is included as the last statement, in a function, to return a value. The function terminates if return statement is executed, or it has executed the last statement in a function.

Example:
```
<?php
average();
function average()
{
        $sub1=50;
        $sub2=50;
        $total = $sub1 + $sub2;
        $avg = $total / 2;
        printf(—The Average is: %f‖, $avg);
}
?>
```

(i) Parameter passing Techniques:

Parameter passing mechanisms are the methods of passing the parameters, to a function. The parameters that are passed in the call to a function are called, actual parameters and the parameters

that are listed in the function definition are called, formal parameters. Actual parameters can be any expression or a variable, but formal parameters must be variables.

An important point is that the number of actual parameters will be unbound variables. On the other hand, if actual parameters are more, then the extra parameters will be ignored. Thus, PHP supports functions with variable number of parameters. There are TWO parameter passing mechanisms. One is *_Call by Value'* and second is *_Call by Reference'*. **a) Call by Value:**

This is the default parameter passing mechanism of PHP. This mechanism copies the values of actual parameters, into the formal parameters. If the called function changes the values of the formal parameters, then there will be no change in the actual parameters because, formal parameters are never copied back to the caller. Thus, call by value mechanism provides one-way communication.

Example:

```
<?php
$price=15.00;
$tax=0.25;
echo "price = $price (before func. call) <br />";
caltax($price,$tax);
function caltax($price, $tax)
{
        $price = $price + ($price * $tax);
        echo "Total cost = $price <br />";
}
echo "price = $price (after func. call)";
?>
```

Output: price = 15 (before func. call)
Total cost = 18.75 price = 15 (after func. call)

b) Call by Reference:

Call by Reference mechanism, provides two way communications by passing the memory address of the actual parameters, rather than values, to the function. Thus, when the called function changes the formal parameters, these changes also affect the actual parameters.

In order to specify call by reference parameters, PHP uses an ampersand sign (&), before the name of the formal parameters that need to be passed by reference. However, the actual parameters must be variables for this method.

Example:

```
<?php
$price=15.00;
$tax=0.25;
echo "price = $price (before func. call) <br />";
caltax(&$price,$tax);
function caltax($price, $tax)
{
        $price = $price + ($price * $tax);
        echo "Total cost = $price <br />";
}
echo "price = $price (after func. call)";
?>
```

Output: price = 15 (before func. call)
Total cost = 18.75
price = 18.75 (after func. call)

c) Default Argument Values

Default values can be assigned to input arguments, which will be automatically assigned to the argument if no other value is provided. Example

```
<?php $a=20;
myFun($a,10);
echo $a,"<br>";
function myFun($a,$c,$value=0.876)
{
$c=$a+$value+$c; echo
"<br> $c <br>";
$value+=10;
$a=$a*$a; echo
$value;
}
?>
```

//output: 30.876, 10.876400

Recursive Functions

Recursive functions, or functions that call themselves.

```
<?php
function factorial ($natural) {
// This guarantees that the function will return 1 (even with 0 as argument)
$result = 1; if
($natural > 0) {
// Here we're applying the second formula: n * (n - 1)!
$result = $natural * factorial(($natural - 1));
}
// return the acumulated result return
$result;
}
echo factorial(6);
?>
```

OUTPUT: 720

3.8 REGULAR EXPRESSIONS

Regular expressions provide the foundation for describing or matching data according to defined syntax rules. A regular expression is nothing more than a pattern of characters itself; matched against a certain parcel of text It is used like validations in forms when taking the input from users. The structure of a POSIX (Portable Operating System Interface for Unix) regular expression is similar to that of a typical arithmetic expression: various elements (operators) are combined to form a more complex expression. The meaning of the combined regular expression elements is what makes them so powerful. You can use the syntax to find not only literal expressions, such as a specific word or

number, but also a multitude of semantically different but syntactically similar strings, such as all HTML tags in a file.

Brackets (**[ ]**) are used to represent a list, or range, of characters to be matched.

- [0 9] matches any decimal digit from 0 through 9.
- [a z] matches any character from lowercase a through lowercase z.
- [A Z] matches any character from uppercase A through uppercase Z.
- [A Za z] matches any character from uppercase A through lowercase z.

3.8.1 Character Classes:

In PHP, We have several predefined character ranges, also known as character classes. Character classes specify an entire range of characters for example, the alphabet or an integer set. Standard classes include the following:

- **[: alpha :]**: Lowercase and uppercase alphabetical characters. This can also be specified as [A-Za-z].
- **[: alnum :]**: Lowercase and uppercase alphabetical characters and numerical digits. This can also be written as [A-Za-z0-9].
- **[: cntrl :]**: Control characters such as tab, escape, or backspace.
- **[: digit :]**: Numerical digits 0 through 9. This can also be specified as [0-9].
- **[: graph :]**: Printable characters found in the range of ASCII 33 to 126.
- **[: lower :]**: Lowercase alphabetical characters. This can also be specified as [a-z].
- **[: punct :]**: Punctuation characters, including ! # $ % ^ & ( ) - + = f g [ ] : ; ' <> , . ? and /.
- **[: upper :]**: Uppercase alphabetical characters. This can also be specified as [A-Z].
- **[: space :]**: Whitespace characters, including the space, horizontal tab, vertical tab, new line, form feed, or carriage return.
- **[: xdigit :]**: Hexadecimal characters. This can also be specified as [a-fA-F0-9].

3.9 PROGRAMMING EXERCISE

1. Write a PHP program to sort list without using sort().

```
<?php
$array=array('2','4','8','5','1','7','6','9','10','3');
echo "Unsorted array is: "; echo
"<br />";
foreach ($array as $value) { echo
$value . ",";
}
echo "<br />"; echo
"<br />";
for($j = 0; $j < count($array); $j ++) {
for($i = 0; $i < count($array)-1; $i ++){
if($array[$i] > $array[$i+1]) {
$temp = $array[$i+1];
$array[$i+1]=$array[$i];
$array[$i]=$temp;
}
```

```
}
}
echo "Sorted Array is: ";
echo "<br />";
#print_r($array); foreach
($array as $value) { echo
$value . ",";
}
?>
```

2. Write a PHP program to decompose a string into individual elements and store them in an array.

Use PHPs explode() function to split a string by delimiter and store the separate segments in a numerically indexed array:

```
<?php
// define string
$alphabetStr = "a b c d e f g h i j k";
// break string into array
// using whitespace as the separator
// result: ("a","b","c","d","e","f","g","h","i","j","k")
print_r(explode(" ", $alphabetStr));
?>
```

3. You want to strip an array of all duplicate elements to obtain a unique set.(Removing Duplicate Elements in an Array)

```
<?php
// define an array containing duplicates
$numbers = array(10,20,10,40,35,80,35,50,55,10,55,30,40,70,50,10,35,85,40,90,30);
// extracts all unique elements into a new array //
result: "10, 20, 40, 35, 80, 50, 55, 30, 70, 85, 90"
echo join(", ", array_unique($numbers));
?>
```

4. Write a PHP program to sort a multidimensional array using multiple keys.

```
<?php
// create a multidimensional array
$data = array();
$data[0] = array("title" => "Net Force", "author" => "Clancy, Tom", "rating"
=> 4);
$data[1] = array("title" => "Every Dead Thing", "author" => "Connolly, John",
"rating"=> 5);
$data[2] = array("title" => "Driven To Extremes", "author" => "Allen,
James", "rating" => 4);
$data[3] = array("title" => "Dark Hollow", "author" => "Connolly,
John", "rating" => 4);
$data[4] = array("title" => "Bombay Ice", "author" => "Forbes,
Leslie", "rating" => 5);
```

```
// separate all the elements with the same key
// into individual arrays foreach
($data as $key=>$value) {
$author[$key] = $value['author'];
$title[$key] = $value['title'];
$rating[$key] = $value['rating'];
}
// sort by rating and then author
array_multisort($rating, $author, $data);
print_r($data); ?>
```

5. Write a PHP script to protect a publicly-displayed e-mail address from being captured by an e-mail address harvester.

```
<?php
// function to protect
// publicly-displayed e-mail addresses
// replace @ with "at"
// . with "dot"
// - with "dash" // _
with "underscore"
function protectEmail($email) {
// define array of search and replacement terms
$search = array(".", "-", "_", "@");
$replace = array(" dot ", " dash ", " underscore ", " at ");
// perform search and replace operation return
str_replace($search, $replace, $email);
}
// result: "dontap at cst dash a dot acm dot org" print
protectEmail("dontap@cst-a.acm.org");
?>
```

3.10 SUMMARY

- Variable is a named memory location that contains data and may be manipulated throughout the execution of the program.
- Variable names begin with a dollar sign (**$**).
- A constant is a value that cannot be modified throughout the execution of a program.
- PHP supports eight primitive data types (Boolean, integer, oat, string, array, object, resource and NULL).
- An integer is representative of any whole number or, in other words, a number that does not contain fractional parts.
- A string is series of characters, where a character is the same as a byte.
- An array is formally defined as an indexed collection of data values.
- An operator is a symbol that specifies a particular action in an expression.
- The result of the modulus operator % has the same sign as the dividend that is, the result of $a % $b will have the same sign as $a.
- Bitwise operators allow evaluation and manipulation of specific bits within an integer.

- The concatenation operator ('.'), which returns the concatenation of its right and left arguments.
- The concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side.
- Looping mechanisms over a simple means for accomplishing a commonplace task in programming: repeating a sequence of instructions until a specific condition is satisfied.
- The foreach construct provides an easy way to iterate over arrays.
- The return statement immediately ends execution of the current function, and returns its argument as the value of the function call.
- The **strip_tags()** function removes HTML tags from a string.
- A function is a named block of code that performs a specific task.

3.11 Questions

1. (a) Explain different types of operators in PHP with examples.
   (b) Write a PHP program that removes HTML from the given string. [Nov-2012,SupR09(IT)] 2.
(a) Explain about PHP data types in detail.
   (b) Write a PHP program that uses all the data types. [Nov-2012 Set-1] 3.
(a) What is ternary operator? Explain with an example.
(b) Write a PHP program that works on date & time using functions.
(c) Explain about scope of a variable. [Nov-2012 Set-2] 4. (a) What is the use of scope resolution operator? Explain.
   (b) Explain about operator precedence & associativity.[Nov-2012 Set-3] 5.
(a) Explain different types of operators in PHP.
   (b) What is the use of scope resolution operator? Explain.[Nov-2012 Set-4] 6.
Explain in detail about statements in PHP.
7. (a) Explain how arrays are used in PHP.
(b) How functions are declare in PHP? Explain recursive functions in PHP with example.
8. (a) List and Explain different array functions in PHP?
(b) Expain different ways to print an array?
9. (a) What is the role of associative arrays in PHP? Explain with examples.
(b) How to declare and access the multidimensional arrays in PHP? Explain.

4.1 <u>Using Cookies</u>

Cookies are a mechanism for storing data in the remote browser .

Then,whenever the browser requests a page on your Web site, all the data in the cookie is automatically sent to the server within the request.This means that you can send the data once to thebrowser, and the data is automatically available to your script from that moment onward. Acookie is sent from the server to the browser as part of the HTTP headers.

4.1.1 Setting Cookies

We can set cookies using the **setcookie()** function.

setcookie() defines a cookie to be sent along with the rest of the HTTP headers. Like otherheaders, cookies must be sent before any output from your script (this is a protocol restriction).This requires that you place calls to this function prior to any output, including < html > and< head > tags as well as any whitespace. ***boolsetcookie(string $name [, string $value [, int $expire = 0 [, string $path [,***

string $domain [, bool $secure = false [, bool $httponly = false]]]]]]) Here is the detail of all the arguments:

- **Name**: - This sets the name of the cookie and is stored in an environment variable calledHTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value**:-This sets the value of the named variable and is the content that you actuallywant 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 cookiewill automatically expire when the Web Browser is closed.
- **Path**:-This specifies the directories for which the cookie is valid. A single forward slashcharacter permits the cookie to be valid for all directories.
- **Domain**:- This can be used to specify the domain name in very large domains and mustcontain at least two periods to be valid. All cookies are only valid for the host and domainwhich 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 regularHTTP.
- **httponly**:- When TRUE the cookie will be made accessible only through the HTTPprotocol. This means that the cookie won't be accessible by scripting languages, such asJavaScript. It has been suggested that this setting can effectively help to reduce identitytheft through XSS attacks (although it is not supported by all browsers), but that claim is often disputed. Added in PHP 4.2.0. TRUE or FALSE.

Example

```
<?php
//File name: setcook.php
//Let's say that the correct login is based on these global user and pass values.
//In the real world, this would be taken from the database most likely.
$username="test_uname"; $password="test_pwd"; setcookie
("cookie_user", $username, time()+60*60*24*30); setcookie
("cookie_pass", $password, time()+60*60*24*30);
header("read_cook.php");
```

?> when we execute above script the cookie values are stored internally in the form of arrays asfollows:

Array ([cookie_user] =>test_uname [cookie_pass] =>test_pwd)

4.1.2 Reading Cookies

Cookies can indeed be read and quite easily. By using the $_COOKIE superglobal, you can have full access to your cookie for reading and writing to it from your script.

The following example demonstrates reading values from cookies. In this example we retrieve values of previous example cookies.

Example

```
<?php
```

```
//File name: read_cook.php echo "User name from cookie:".$_COOKIE['cookie_user']; echo "<br>password from cookie:".$_COOKIE['cookie_pass'];
?>
```

:::::: OUTPUT::::::

User name from cookie:test_uname password
from cookie:test_pwd

4.1.3 Deleting Cookies

Removing cookies is also a simple task.

- You should note that cookies will disappear by themselves if you have set them up to doso. Example is as follows:

```
setcookie("cookie_user", $value, time()+3600); /* expire in 1 hour */
```

- Cookies that have not been assigned a time to die will simply be removed when the browser window closes. Example is as follows:

```
setcookie("cookie_user", $value);
```

- A user will want to be able to clear the cookies on a site. When deleting a cookie you should assure that the expiration date is in the past, to trigger the removal mechanism in your browser. Examples follow how to delete cookies sent in previous example: <*?php*

```
// set the expiration date to one hour ago
setcookie ("cookie_user", "", time() - 3600);
?>
```

::::: OUTPUT ::::::

After execution of above script it returns null value.

4.2 Using HTTP Headers

HTTP headers are powerful sets of functionality supported by PHP. The most important aspect to remember about headers is that they can be called only before any output has been written to the web page. We can use them to control everything, including setting the current page location, finding out what file format is being displayed, and managing all aspects of the browser cache. In the following examples, you will learn how to use the **header()** function in a variety of ways.

Prototype of **header()** function is as follows: ***void header (string $string [, bool $replace = true [, int $http_response_code]])*** **sring:** The header string. **replace:** The optional replace parameter indicates whether the header should replace a previous similar header, or add a second header of the same type. By default it will replace, but if you pass in FALSE as the second argument you can force multiple headers of the same type.

http response code: Forces the HTTP response code to the specified value. Note that this parameter only has an effect if the string is not empty.

4.2.1 Redirecting to a Different Location

One of the more common uses for HTTP headers is redirecting a script. By using headers inside processing scripts, you can force the browser to return to any page we want.

The following program shows the login page. After successful login **header()** function forwarded to **home.php** page.

login.php

```
<html>
<title>Login</title>
```

```
<style> .error {
font-weight: bold;
color: #FF0000;
}
</style>
</head>
<body>
<div style="width: 500px; text-align: left;">
<?php
$host="localhost"; // Host name
$username="root"; // Mysql username
$password=""; // Mysql password
$db_name="cse"; // Database name
$tbl_name="reg";  //  Table  name
//Default to showing the form.
$flag = false;
//Handle the incoming data.
if ($_SERVER["REQUEST_METHOD"] == "POST"){
//Let's declare a submission value that tells you if you are fine.
$flag = true; //Validate the name. if (trim ($_POST['name'])
== "" || trim ($_POST['pwd'])==""){
$flag = false;
echo ("<p class=error>Sorry, you must enter your name & Password.</p><br />");
} else{
mysql_connect("$host", "$username", "$password")or die("cannot connect");
mysql_select_db("$db_name")or die("cannot select DB");
$sql="SELECT * FROM $tbl_name where name='".$_POST['name']."' and
pwd='".$_POST['pwd']."'"; $result=mysql_query($sql);
if($info = mysql_fetch_array($result)){
$flag = false;
header("Location: home.php"); //this function forward to home.php }else{
$flag = false;
echo ("<p class=error>Sorry, username or/and password is/are wrong.</p><br />");
}
} }
if (!$flag){
?>
<form action="<?php $_SERVER["PHP_SELF"];?>" method="post">
<p>Login form:</p>
User Name: <input type="text" name="name" maxlength="150" value="<?php
if ($_SERVER["REQUEST_METHOD"] == "POST"){echo $_POST['name'];}?>"/><br /><br />
Password: <input type="password" name="pwd" maxlength="150" value="<?php if
($_SERVER["REQUEST_METHOD"] == "POST"){echo $_POST['pwd'];}?>" /><br /> <input
type="submit" value="Submit" style="margin-top: 10px;" />
```

```
</form>
<?php
}
?>
</div>
</body> </html>
```

home.php

```
<html>
<head><title> header( ) example</title></head>
<body>
<h1>This is the home page <br>after successful login header function forwarded to this page.</h1>
</body>
</html>
```

:::::::::: OUTPUT ::::::::::

This is the home page after successful login header function forwarded to this page.

The **header()** function is rather nice in that it will redirect you automatically to the appropriate file without a single interruption in the processing. You will simply find yourself at the appropriatepage.

4.2.2 Sending Content Types Other Than HTML

The Content-Type header field is used to specify the nature of the data in the body of an entity, by giving type and subtype identifiers, and by providing auxiliary information that maybe required for certain types. After the type and subtype names, the remainder of the header field is simply a set of parameters, specified in an attribute/value notation. The set of meaningful parameters differs for the different types. The ordering of parameters is not significant.

The header function is more than versatile enough to take care of this issue. To make the most out of this function, you can effectively output other file types by simply declaring the content type you want to output.

Content Type	Application
application/pdf	Adobe Portable Document Format (PDF) types
application/msword	Microsoft Word documents
application/excel	Microsoft Excel documents
image/gif	GIF images
image/png	PNG images
application/octetstream	Zip files
text/plain	Plain text (text files)

Table 4.1: Common File Format Content Types

4.2.3 Forcing File "Save As" Downloads

Because web browsers can output many different file types directly onto the screen, the defaultwhen you use headers to output a wide variety of file types is to make them automaticallyappear on the screen. What if you would rather have the file appear as a download, though?You can use the header()function to force a Save As dialog box to appear for the user to accepta download.

```
<?php
header("Content-type:application/pdf");
$output=downloaded;
// It will be called downloaded.pdf
header("Content-Disposition:attachment;filename=".$output.".pdf");
// The PDF source is in original.pdf readfile("original.pdf");
?>
```

The key point in this code is showing *content-disposition* in the header. By making *contentdisposition* an attachment value, the browser will force a download rather than display the file inline. By using this, you can force the download to appear with any particular filenameyou prefer and also with pretty much any file extension. By using content-type, you force thebrowser to output a file of the requested type.

4.3 Using Sessions

Because cookies are getting less and less trusted, a means had to be created to allow user authentication without having to store physical data on a client system. As a solution, sessions came onto the scene. **session_start()** function needs to be called at the beginning of every page where you want session access. When session_start() is called or when a session auto starts, PHP will call the open and read session save handlers. This function returns TRUE if a session was successfully started, otherwise FALSE.

The prototypes for these session-related functions are as follows: ***boolsession_start (void)***

boolsession_destroy (void)

4.3.1 Setting Sessions

Setting the data in sessions is very easy in php. The session support allows to store data between requests in the $_SESSION superglobal array. The following example creates a session state, sets a session:

```
<?php
session_start();
$_SESSION['uname']="test_user";
$_SESSION['pwd']="test_pwd";
?>
```

When we are executing the above script internally SESSION store the information in an associative array as follows:

Array ([uname] =>test_user [pwd] =>test_pwd)

4.3.2 Reading Sessions

Reading the values from SESSIONs is also very easy using $_SESSION superglobal variable. The following example shows the reading and accessing data from SESSIONs.

```
<?php
session_start();
echo "Username in SESSION is <b>".$_SESSION['uname']."</b><br>";
echo "Password in SESSION is <b>".$_SESSION['pwd']."</b>"; ?>
```

:::::: OUTPUT ::::::

Username in SESSION is **test_user**
Password in SESSION is **test_pwd**

4.3.3 Deleting Sessions

In php we are using three functions to delete or clear SESSION information.

- session_unset()

The session_unset() function frees all session variables currently registered. the prototype is as follows:

voidsession_unset (void)

session_destroy()

The session_destroy() destroys all of the data associated with the current session. It does not unset any of the global variables associated with the session, or unset the session cookie. To use the session variables again, session_start() has to be called. The prototype is as follows: ***bool session destroy (void)***

- unset() unset() destroys the specified variables. The behavior of unset() inside of a function canvary depending on what type of variable you are attempting to destroy.

If a globalized variable is unset() inside of a function, only the local variable is destroyed.The variable in the calling environment will retain the same value as before unset() wascalled. The prototype is as follows: ***void unset (mixed $var [, mixed $...])***

Example: unset($_SESSION[`uname']);

After executing the above statement only _uname‘ will be cleared from session.

4.3.4 Storing Simple Data Types in Sessions:

Sessions have really been useful only for passing simple data types around. Sessions handlesimple data types, and they handle them well. Like any PHP variable, however, the data typeof a current session is based upon what was last assigned to it and can be changed quite easily. The following example describes this:

```
<?php
session_start();
(int)$_SESSION['inumber']=1024;
(float)$_SESSION['fnumber']=10.23;
(string)$_SESSION['str']="String";
?>
```

Internally SESSION stores information as follows:

Array ([inumber] => 1024 [fnumber] => 10.23 [str] => String) **4.3.5**

Storing Complex Data Types in Sessions:

One of the major improvements to PHP is the ability to store complex data types (arrays andobjects) within a session.

Store & Access Arrays in SESSION

The following example demonstrates how to store arrays in Sessions.

```
<?php
session_start();
$x=array('uname'=> "suresh",'pwd'=>"1234");
$_SESSION['str']=$x;
?>
```

Internally session store the data in the form of two-dimensional array as follows:

Array ([str] => Array ([uname] => Praveen [pwd] => password))

The following example demonstrates accessing the arrays in sessions. When we store arrays in SESSIONs we are accessing $_SESSION as a two dimensional array.

```
<?php
session_start();
echo "User name::::::::::".$_SESSION['str']['uname']."<BR>"; echo
"Password::::::::::".$_SESSION['str']['pwd']."<BR>";
?>
```

:::::: OUTPUT ::::::

User name::::::::::suresh

Password::::::::::1234

Store & Access Objects in SESSION

PHP allows you to store objects within sessions. Using this technique, you can easily storelarge quantities of data within a single object, use the functionality within the session for thesepurposes, and then pass the data along to otherpages.

```
<?php
session_start();
//A class that does not do too much.
classmyclass { protected $myvalue;
public function setmyvalue ($newvalue){
$this->myvalue = $newvalue;
}
public function getmyvalue (){
return $this->myvalue;
}
}
$_SESSION['myclass_value'] = new myclass ();
//This function exists for the sole purpose of showing how sessions can be called
//from anywhere within the scope of the session state. functionoutputsessions (){
$_SESSION['myclass_value']->setmyvalue ("Hello World"); echo
$_SESSION['myclass_value']->getmyvalue ();
}
//Then you can call the function from here: outputsessions();
?>
```

Output:

Hello World

Difference between cookies & sessions

	Cookies	**Sessions**
1	Cookies store data on the Client machine	Sessions store data on the server
2	Less secure	More Secure than cookies
3	Stores less amount of data	Here we store more amount of data
4	It store only primitive datatypes	It stores both primitive and Compounddatatypes.
5	More amount of time Data stored in cookies	Less time sessions store the data.

6	No need to require additional space tostore data	Sessions need extra space, unlike cookiesto store data
7	No need to call additional functions	Here it needssession_start() to create sessions.
8	Setting and deleting use setcookie()	For setting session use $_SESSION[] superglobal array, and delete session information use unset() or session_destroy().
9	Accessing cookie using $_COOKIE[] superglobal array.	Accessing sessions using $_SESSION[] superglobal array

Table 4.1: Difference between cookies & sessions

4.4 Authenticating Your Users

PHP uses two predefined variables to authenticate a user.

1. $_SERVER['PHP_AUTH_USER']
2. $_SERVER['PHP_AUT_PW']

These two superglobal variables store the Username and password values respectively whileauthenticating is as simple as comparing the expected username and password to these variables. The isset() function determines whether a variable has been assigned a value. The prototypeis as follows: ***booleanisset(mixed var[,mixed var[,.....]])***

It returns TRUE if the variable contains a value and FALSE if it does not.It applied to User authentication, the isset() function is useful for determining whether the $_SERVER['PHP_AUTH_USER'] and $_SERVER['PHP_AUTH_PW'] variables are properly set.

Using isset() to verify whether a variable contain a value or not.

```
<?php
if(!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])){
header('www-Authenticate: Basic replays="Authentication"'); header("HTTP/1.1
401 Unauthorized");
} else{
echo "User Name is ".$_SERVER['PHP_AUTH_USER']."<br/>"; echo
"password is ".$_SERVER['PHP_AUTH_PW'];
}
?>
```

4.5 Using Environment and Configuration Variables

PHP provides a means to use and verify the configuration settings and environment variablesrelative to the server space the script is occupying. A common use of the environment variablesin PHP is for dynamic imaging. While Windows systems commonly store their fonts in onefolder, Linux-based systems keep theirs in another. By using PHPs environment variables todetermine the current operating system, you can make your code slightly more portable.

Using configuration variables can also come in quite handy, particularly with file uploadscripts. The base PHP installation leaves only enough processing time to upload files that aregenerally 2MB or smaller in size. By manipulating the PHP configuration files temporarily, youcan increase the limit enough to allow a script to process much larger files.

4.5.1 Reading Environment Variables

The $_ENV superglobal is PHPs method for reading a systems environment variables and hasan argument set that is based upon the current environment that is available to it. We canretrieve them using the getenv() function also.

*<?php echo $_ENV['Program Files'] . "
"; //Outputs C:\Program Files.*
*echogetenv("ProgramFiles") . "
"; //Outputs C:\Program Files. echo*
*$_ENV['COMPUTERNAME'] . "
"; //Outputs COMPUTER-2339.*
*echogetenv("COMPUTERNAME") . "
"; //Also Outputs COMPUTER-2339.*
?>

4.5.2 Reading Configuration Variables

Reading configuration variables, on the other hand, takes place through two functions, ini_get() and ini_get_all(). The function ini_get() will retrieve the value of a specified configuration variable, and the function ini_get_all() will retrieve an array filled with the entire selection ofconfiguration variables that are available.

*<?php echoini_get ("post_max_size") . "
";*
//Outputs 8MB.
//And you can output the entire listing with this function. print_r
(ini_get_all());
?>

4.5.3 Setting Environment Variables

Setting environment and configuration variables is just as easy as it is to get them. Whileworking with environment variables, you merely need to assign a new value to the $_ENVsuperglobal to process a temporary change. The change will be in effect for the scripts duration.

*<?php echo $_ENV['COMPUTERNAME'] . "
"; // Echoes*
COMPUTER-2339.
$_ENV['COMPUTERNAME'] = "dp"; echo $_ENV['COMPUTERNAME'] . "<br
/>"; //Echoes the new COMPUTERNAME.
?>

4.5.4 Setting Configuration Variables

The same applies for configuration variables but with a different approach. To set a configurationvariable, you have to use the PHP function ini_set(), which will allow you to set a configurationvariable for the scripts duration. Once the script finishes executing, the configuration variablewill return to its original state. The prototype for ini_set() is as follows: ***string ini_set (string varname, string newvalue)***

<?php echoini_get ('post_max_size');
//Echoes 8MB.
//Then you set it to 200M for the duration of the script.
ini_set('post_max_size','200M'); echoini_get
('post_max_size'); //Echoes 200MB.
//Any files that are to be uploaded in this script will be OK up to 200M. ?>

4.6 Working with Date and Time

In the programming world, date and time values formatted in Unix epoch manner.The Unix epoch (or Unix time or POSIX time or Unix timestamp) is the numberof seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leapseconds (in ISO 8601: 1970-0101T00:00:00Z). Literally speaking the epoch is Unix time 0(midnight 1/1/1970), but 'epoch' is often used as a synonym for 'Unix time'. Many Unixsystems store epoch dates as a signed 32-bit integer, which might cause problems on January19, 2038 (known as the Year 2038 problem or Y2038). The following are some of the date and time functions.

4.6.1 time()

It returns the current system date and time as a Unix timestamp.

```
<?php
echo time();
?>
```

:::::: OUTPUT ::::::

1386331271

4.6.2 date()

Returns a string formatted according to the given format string using the given integer timestamp or the current time if no timestamp is given. The prototype is a s follows: ***string date (string $format [, int $timestamp = time()])***

The following example demonstrate date() function.

```
<?php
echo date("r");
echo "<br>".date("d:M:Y:::H:S:I");
?>
```

:::::: OUTPUT ::::::

Fri, 06 Dec 2013 13:24:49 +0100

06:Dec:2013:::13:49:0

Character	Description
F	Full name of the month (January, February, and so on). M Three-letter abbreviation for the month (Jan, Feb, and so on).
m	Numeric representation for the month, with leading zero (two digits).
n	Numeric representation for the month (no leading zero). y
Two-digit year. Y	Four-digit year.
d	Day of the month, with leading zeros (two digits). j Day of the month (no leading zeros).
D	A textual representation of a day, three letters (Mon, Tue, and so on).
w	Numeric representation of the day of the week (0 = Sunday, 6 = Saturday). h Hour in 12-hour format, with leading zero (two digits).
g	Hour in 12-hour format (no leading zero).
H	Hour in 24-hour format, with leading zero (two digits).
G	Hour in 24-hour format (no leading zero).
a	am/pm (lowercase). A
AM/PM (uppercase). i	Minute, with leading zero (two digits). j Minute (no leading zero) s Second, with leading zero (two digits).
r	RFC-2822 format WWW, DD MMM YYYY HH:MM:SS HHMM

Table 4.2: Formatting Characters for the date() Function

4.6.3 checkdate()

It checks the validity of the date formed by the arguments. A date is considered valid if eachparameter is properly defined. The prototype is as follows: ***boolcheckdate (int $month , int $day , int $year)***
Returns TRUE if the date given is valid; otherwise returns FALSE.

```
<?php
echo checkdate(12, 31, 2000);// Return 1
echo checkdate(2, 29, 2001); //Return 0
?>
```

4.6.4 mktime()

Returns the Unix timestamp corresponding to the arguments given. This timestamp is a longinteger containing the number of seconds between the Unix Epoch (January 1 1970 00:00:00GMT) and the time specified.

Arguments may be left out in order from right to left; any arguments thus omitted will beset to the current value according to the local date and time. The prototype is as follows:

intmktime ([int $hour = date("H") [, int $minute = date("i") [, int $second =date("s") [, int $month = date("n") [, int $day = date("j") [, int $year = date("Y")[, int $is_dst = -1]]]]]]) A simple example is:

```
<?php
echomktime(11,00,6,12,6,2012);//Output: 1354788006
?>
```

4.6.5 strtotime()

The function expects to be given a string containing an English date format and will try toparse that format into a Unix timestamp (the number of seconds since January 1 1970 00:00:00UTC), relative to the timestamp given in now, or the current time if now is not supplied.

Each parameter of this function uses the default time zone unless a time zone is specifiedin that parameter. Be careful not to use different time zones in each parameter unless that isintended. The prototype is as follows: ***intstrtotime (string $time [, int $now = time()])***

Example to demonstrate strtotome()

```
<?php
echo strtotime("20131206"); // Output: 1386284400
?>
```

4.6.6 getdate()

It returns an associative array containing the date information of the timestamp, or the currentlocal time if no timestamp is given. ***arraygetdate ([int $timestamp = time()])***

Example programs demonstrate getdate()

```
<?php
print_r(getdate(time()));
?>
```

:::::: OUTPUT ::::::

Array ([seconds] => 36 [minutes] => 4 [hours] => 14 [mday] => 6 [wday] => 5[mon] => 12 [year] => 2013 [yday] => 339 [weekday] => Friday[month] => December [0] => 1386335076)

4.6.7 strptime() strptime() returns an array with the date parsed, or
FALSE on error. ***array strptime (string $date , string $format)***

Function	Description
date_sunrise()	Returns time of sunrise for a given day and location (new in PHP 5).
date_sunset()	Returns time of sunset for a given day and location (new in PHP 5).
gmdate()	Formats a GMT/UTC date/time. Uses the same formatting charactersas the date() function.
gmmktime()	Converts a set of GMT date/time values into a Unix timestamp (analogous to mktime()).

Table 4.3: More PHP 5 Date/Time Functions

4.7 Programming Exercise

1. Write a PHP program that works on date & time using functions.

```
<?php
$b = time ();
print date("m/d/y",$b) . "<br>";
print date("D, F jS",$b) . "<br>";
print date("l, F jSY",$b) . "<br>";
print date("g:iA",$b) . "<br>"; print
date("r",$b) . "<br>";
print date("g:i:s A D, F jSY",$b) . "<br>";
?>
```

4.8 SUMMARY

- **Cookies** are a mechanism for storing data in the remote browser.
- A cookie lets you store a small amount of data.
- We can set and delete cookies using the **setcookie()** function.
- Arguments we pass through setcookie() function are: **name, expiry, value, path, domain**.
- By using the **$_COOKIE**superglobal, you can havefull access to your cookie for readingand writing to it from your script.
- By using **header()** function, we access the HTTP headers in PHP.
- We access HTTP headers in three ways:
    1. Redirecting to a Different Location
    2. Sending Content Types Other Than HTML
    3. Forcing File Save As Downloads
- **session_start()** function needs to be called at the beginning of every page where you wantsession.
- The session support allows to store data between requests in the **$_SESSION**superglobalarray access.
- Deleting the SESSION information using three functions: **session_unset()**, **session_destroy()**and **unset()**.
- The **session_unset()** function frees all session variables currently registered.
- The **session_destroy()** destroys all of the data associated with the current session.
- The **unset()** destroys the specified variables.
- PHP has the ability to store complex data types (arrays and objects) within a session.
- **$_ENV**superglobal variable or **getenv()**is used to read Environment variables.

- Reading configuration variables, on the other hand, takes place through two functions, **ini_get()** and **ini_get_all()**.
- To set the Configuration variables use the function **ini_set()**.
- In the programming world, date and time values formatted in **Unix epoch** manner.
- **time()** function returns the current system date and time as a Unix timestamp.
- **checkdate()** function is used to validate dates.

4.9Questions

1. (a) How to set a cookie on user computer? Explain with an example.
(b) Explain Briefly how to redirect the HTTP headers to different locations. *[Apr/May 2012 Set1]*
2. (a) What are cookies? What are the advantages of cookies?
(b) Briefly explain different parameters available when setting a cookie. *[Apr/May 2012 Set-2][IT-Nov/Dec2012 Set-(R09)]*
3. What are Cookies? Explain the following:
(a) Setting Cookies. (b) Deleting Cookies. *[Apr/May 2012 Set-3]* 4. (a) Explain briefly how to use the header() function in different ways. (b) Explain the advantages of the PHP functions available for the time and date. *[Apr/May 2012 Set-4]*
5. (a) Explain briefly how to redirect the HTTP headers to different locations.
(b) Explain briefly how to use the header ( ) function in different ways. *[Nov 2012 Set-1][5(b) Nov 2012 Set-3]*
6. (a) Explain why cookies are becoming less trusted. *[5(a) Nov 2012 Set-3]* (b) What is a session? Explain briefly about sessions. *[Nov 2012 Set-2]* 7. (a) What are the advantages and disadvantages of cookies?
(b) How to set a cookie on user computer? Explain with an example.*[Nov 2012 Set-4]* 8.
(a) What is a session? Explain briefly about sessions.
(b) Explain with an example, how storing of simple data types is done in sessions. *[jan-2014 set-1]*
9. What are cookies? What are the advantages and disadvantages of cookies? *[jan-2014 set-3]* 10. Explain the following functions with examples:
(a) getdate() (b) microtime() (c) strptime() (d) idate() *[jan-2014 set-2]*
11. Explain the following functions with examples.
(a) date_sunrise() (b) gmmktime() (c) strftime() (d) time() *[jan-2014 set-4]*

Cookies:

A cookie is a small piece of text file stored on user's computer in the form of name-value pair. Cookies are used by websites to keep track of visitors e.g. to keep user information like username etc. If any web application using cookies, Server send cookies and client browser will store it. The browser then returns the cookie to the server at the next time the page is requested. The most common example of using a cookie is to store User information, User preferences, Password Remember Option etc. It is also one of the common and mostly asked interview questions.
Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. PHP transparently supports HTTP cookies.

A cookie lets you store a small amount of data within the user's browser itself. Then, whenever the browser requests a page on your Web site, all the data in the cookie is automatically sent to the server within the request. This means that you can send the data once to the browser, and the data is automatically available to your script from that moment onward. A cookie is sent from the server to the browser as part of the HTTP headers.

We can set cookies using the **setcookie()** function. Cookies can indeed be read and quite easily. By using the $_COOKIE superglobal, you can have full access to your cookie for reading and writing to it from your script.

Some facts about Cookie

1. · Cookies are domain specific i.e. a domain cannot read or write to a cookie created by another domain. This is done by the browser for security purpose.
2. · Cookies are browser specific. Each browser stores the cookies in a different location. The cookies are browser specific

and so a cookie created in one browser (e.g in Google Chrome) will not be accessed by another browser (Internet

Explorer/ Firefox).

3. · Most of the browsers store cookies in text files in clear text. So it‘s not secure at all and no sensitive information should be stored in cookies.
4. · Most of the browsers have restrictions on the length of the text stored in cookies. It is 4096(4kb) in general but could vary from browser to browser.
5. · Some browsers limit the number of cookies stored by each domain (20 cookies). If the limit is exceeded, the new cookies will replace the old cookies.
6. · Cookies can be disabled by the user using the browser properties. So unless you have control over the cookie settings of the users(for e.g. intranet application), cookies should not be used.
7. · Cookie names are case-sensitive. E.g. UserName is different than username.

Advantages of using cookies

1. · Cookies are simple to use and implement.
2. · Occupies less memory, do not require any server resources and are stored on the user's computer so no extra burden on server.
3. · We can configure cookies to expire when the browser session ends (session cookies) or they can exist for a specified length of time on the client‘s computer (persistent cookies). 4. · Cookies persist a much longer period of time than Session state.
5. · No need to call additional functions

Disadvantages of using cookies Here

are some of the disadvantages:

1. · Cookies are not secure as they are stored in clear text they may pose a possible security risk 2. · Several limitations exist on the size of the cookie text (4kb in general), number of cookies (20 per site in general), etc. 3. · User has the option of disabling cookies on his computer from browser‘s setting.
4. · Cookies will not work if the security level is set to high in the browser.
5. · Users can delete a cookie.
6. · Users browser can refuse cookies, so your code has to anticipate that possibility.
7. · Complex type of data not allowed (e.g. dataset etc). It allows only plain text (i.e. cookie allows only string content)

Unit-4

Creating and Using Forms Understanding Common Form Issues, GET vs. POST, Validating form input, Working with multiple forms, and Preventing Multiple Submissions of a form.
XML: Basic XML- Document Type Definition XML Schema DOM and Presenting XML, XML Parsers and Validation, XSL and XSLT Transformation, News Feed (RSS and ATOM).

****

To create a fully functional web application, you need to be able to interact with your users. The common way to receive information from web users is through a form. Web forms are only Hypertext Markup Language (HTML) elements. PHP 5 is built so that it seamlessly integrates with form elements. Over the past few versions of PHP, its methodology for dealing with form information has gradually evolved and is now quite robust.

1. Understanding Common Form Issues

When dealing with forms, the most important aspect to remember is that you are limited to a certain variety of fields that can be applied to a form. The fields that have been created are non-negotiable and work in only the way they were created to work. The < form > element bundles together all the form widgets (also known as controls or fields).

Various Form Elements and their Description

* **TEXT INPUT** - A simple text box
* **PASSWORD INPUT** - A text box that hides the characters inputted
* **HIDDEN INPUT** – A field that does not show on the form but can contain data
* **SELECT** - A drop-down box with options
* **LIST** - A select box that can have multiple options selected
* **CHECKBOX** - A box that can be checked
* **RADIO** - A radio button that can act as a choice
* **TEXTAREA** - A larger box that can contain paragraph-style entries
* **FILE** - An element that allows you to browse your computer for a file
* **SUBMIT** - A button that will submit the form
* **RESET** - A button that will reset the form to its original state

A well-designed form, divides itself into logical chunks using the < fieldset > element. Each chunk gets a title, courtesy of the < legend > element. The following Example demonstrate the forms

registration.html

```
<html>
<head>
<title>Registration form</title>
</head>
<body>
<h1>Registration Form</h1>
<form name=—registration‖ method=—‖ action=‖>
<fieldset>
<legend>Personal Information</legend>
Name : <input type='text' name='name' placeholder="Name As Per SSC"><br>
Father Name: <input type=—text‖ name=—fname‖ placeholder=—Name As Per SSC‖ ><br>
Gender: <select id="gender">
<option value="female">Female</option>
<option value="male">Male</option>
</select><br>
Age: <input id="age" type="number" min="0" max="120"><br> </fieldset> <fieldset>
<legend>Contact Details</legend>
Telephone: <input id="telephone"><br>
Email: <input type='text' id="email"><br>
</fieldset>
<fieldset>
<legend>Pick Your Favorite Animals</legend>
<input id=—cow" type="checkbox"> Cow
<input id="cat" type="checkbox"> Cat <input
id=—dog" type="checkbox"> Dog
<input id="elephant" type="checkbox"> Elephant
</fieldset>
<p><input type="submit" value="Submit Application"></p>
</form>
</body>
</html>
```

6.2 GET vs. POST

The two ways available to a web developer that the information entered into the form is transmitted to its destination by using method. the two methods are GET and POST.

GET

When sending data using the GET method, all fields are appended to the Uniform Resource Locator (URL) of the browser and sent along with the address as data. Sending data using the GET method means that fields are generally capped at 150 characters, which is certainly not the most effective means of passing information. It is also not a secure means of passing data, because many people know how to send information to a script using an address bar. PHP's current methods for dealing with GET variable is the $_GET superglobal. Syntax is **$_GET['Variable Name'];**.

The following Example demonstrate the working of the GET method

File Name: **get.php**

```
<html>
<head>
<title>Example for get</title>
</head>
<body>
<form action="get-demo.php" method="GET">
<p>GET Example:</p>
User Name: <input type="text" name="uname" maxlength="150" /><br /><br />
Password: <input type="password" name="pwd" maxlength="150" /><br />
<input type="submit" value="Submit with GET" style="margin-top: 10px;" /> </form>
</body>
</html>
```

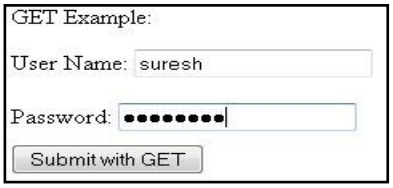


Fig: Output of get.php

File Name: **get-demo.php**

```
<html>
<head>
<title>Example for GET</title>
</head>
<body> <?php if (trim ($_GET['uname']) != "" && trim
($_GET['pwd']) != ""){ echo "Your User Name (with GET): " .
$_GET['uname']; echo "<br>Your password (with GET) : ".
$_GET['pwd'];
} else {
echo "You must submit a value.";
}
?><br /><a href="get.php">Try Again</a>
</body>
</html>
```

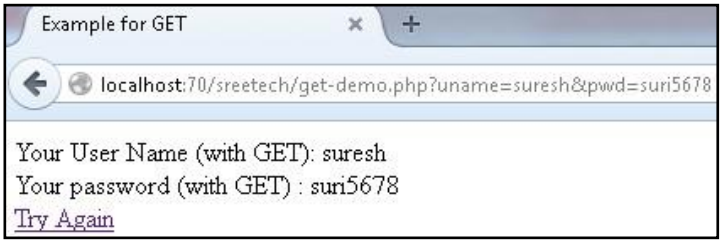


Fig: Output after submit get.php (see the address bar)

When using the GET method, hitting the Refresh button after submitting data the browser will automatically send the data again.

POST

When sending data using the POST method, values are sent as standard input (the data will be sended through body not in URL). Sending data using the POST method is quite a bit more secure (because the method cannot be altered by appending information to the address bar) and can contain as much information as you choose to send. Therefore, whenever possible, use the POST method for sending information and then adjust your script to handle it.

PHP s current methods for dealing with POST variable is the **$_POST** superglobal. Syntax is **$_POST['Variable Name']**;.

The following Example demonstrate the working of the POST method

File Name: **post.php**

```
<html>
<head>
<title>Example for POST</title>
</head>
<body>
<form action="post-demo.php" method="post">
<p>POST Example:</p>
<input type="hidden" name="submitted" value="yes" />
User Name: <input type="text" name="uname" maxlength="150" /><br /><br />
Password: <input type="password" name="pwd" maxlength="150" /><br />
<input type="submit" value="Submit with POST" style="margin-top: 10px;" /> </form>
</body>
</html>
```

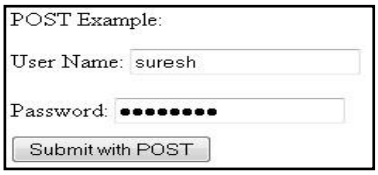


Fig: Output of post.php

File Name: **post-demo.php**

```
<html>
<head>
<title>Example for POST</title>
</head>
<body> <?php if (trim ($_POST['uname']) != "" && trim
($_POST['pwd']) != ""){ echo "Your User Name (with POST): " .
$_POST['uname']; echo "<br> Your password(with POST) :".
$_POST['pwd'] ;
} else {
echo "You must submit a value."; }
?><br /><a href=—post.php">Try Again</a>
</body>
</html>
```

Fig: Output after submit post.php (see the address bar)

Hitting the Refresh button after submitting data using the POST form, the browser will ask you if you want to resubmit the data that was passed to it previously. If you want to resend the data, you must select Yes (Resend) to this option.

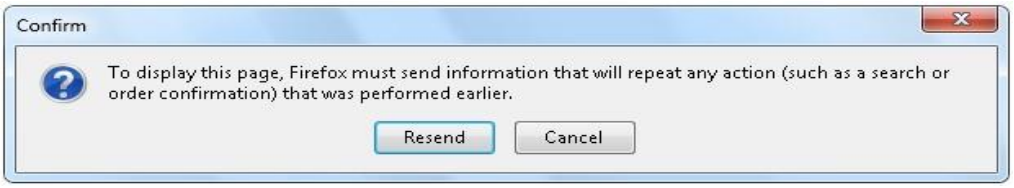


Fig: Refresh after submitting the form it asks confirmation

GET Vs POST

	GET	POST
1	Parameters remain in browser history because they are part of the URL.	Parameters are not saved in browser history.
2	Can be bookmarked.	Can not be bookmarked.
3	GET requests are re-executed but may not be re-submitted to server if the HTML is stored in the browser cache.	The browser usually alerts the user that data will need to be re-submitted.
4	Easier to hack for script kiddies	More difficult to hack
5	Only ASCII characters allowed.	No restrictions. Binary data is also allowed.
6	GET is less secure compared to POST because data sent is part of the URL. So it's saved in browser history and server logs in plaintext.	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs.
7	Restrictions on form data length	No restrictions on form length.
8	GET method should not be used when sending passwords or other sensitive information.	POST method used when sending passwords or other sensitive information.

3 <u>Validating form input</u>

Validation is a way to catch mistakes when they happen (or even better, to prevent them from happening at all).

Client-side validation: These are the checks that happen in the browser, before a form is submitted. The goal here is to make life easier for the people filling out the form.

Examples: HTML5, JavaScript etc.

Server-side validation: These are the checks that happen after a form is sent back to the web server. At this point, it is up to your server-side code to review the details and make sure everything is proper before continuing. No matter what the browser does, server-side validation is essential.

The following example shows a few examples of form validation using PHP.

Filename: **validation.php**

```
<html>
<head>
```

```
<title> Validation DEMO </title>
<?php
if($_SERVER["REQUEST_METHOD"]=="POST"){
if($_POST["uname"]==—‖){
echo "<font color=red>Please Enter valid User name</font><br>";
}
else if(strlen($_POST["uname"])<6){
echo "<font color=red>Please Enter valid User name with more than 6 characters<br></font>";
}
if($_POST["pwd"]==—‖){
echo "<font color=red>Please Enter valid Password</font><br>";
}
else if(strlen($_POST["pwd"])<6){
echo "<font color=red>Please Enter valid Password with more than 6 characters<br></font>";
}
}
?>
</head>
<body>
<form method=POST action="<?php $_SERVER['PHP_SELF']?>" > <table>
<tr> <td>NAME:</td> <td><input type=text name="uname" /></td> </tr>
<tr> <td>PASSWORD:</td> <td><input type=password name="pwd" /></td> </tr>
<tr><td><input type=reset value=CLEAR /></td> <td><input type=submit value=NEXT
/></td></tr>
</table>
</form> <?php
if($_SERVER["REQUEST_METHOD"]=="POST"){ if($_POST["uname"]!=—‖ &&
strlen($_POST["uname"])>=6 && $_POST["pwd"]!=—‖
&& strlen($_POST["pwd"])>=6){ echo "Name:<font
color=green>".$_POST['uname']."<br></font>"; echo "Password:
<font color=green>".$_POST['pwd']."<br></font>";
}
}
?>
</body>
</html>
```

Fig: Output of validation.php after submitting empty values

In the above script: **$_SERVER[“PHP_SELF"]** The filename of the currently executing script, relative to the document root. The above program demonstrates the validation in same page. It is possible to perform the validations using GET and POST methods into the other pages.

4 Working with multiple forms

Sometimes you will need to collect values from more than one page. Most developers do this for the sake of clarity. By providing forms on more than one page, you can separate blocks of information and thus create a flexible experience for the user. The problem, therefore, is how to GET values from each page onto the next page and finally to the processing script.

Being the great developer that you are, you can solve this problem and use the **hidden** input form type. When each page loads, you only load the values from the previous pages into hidden form elements and submit them.

page1.php

```
<html>
<head>
<title>Personal information</title>
</head>
<body>
<form method=POST action="page2.php">
<table align=center>
<tr> <td>NAME</td><td><input type="text" name="name"></td> </tr>
<tr> <td>FATHER NAME</td><td><input type="text" name="fname"></td> </tr>
<tr> <td>MOTHER NAME</td><td><input type="text" name="mname"></td> </tr>
<tr> <td>GENDER</td><td> <input type="radio" name="gen" Value="MALE">MALE
<input type="radio" name=gen value=Female>FEMALE</td> </tr>
<tr> <td><input type=reset value=clear></td> <td><input type="submit" value="NEXT>>"></td>
</tr>
</form>
</body>
</html>
```

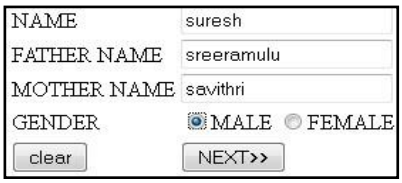


Fig: Output of page1.php

page2.php

```
<html>
<head>
<title>Contact information</title>
</head>
<body>
<form method=POST action="page3.php">
<table align=center>
<tr> <td>E-Mail</td><td><input type="text" name="email"></td> </tr>
<tr> <td>Mobile</td><td><input type="text" name="Mobile"></td> </tr>
<tr> <td>ADDRESS</td><td><textarea name=address></textarea></td> </tr>
```

```
<input type=hidden name="name" value="<?php echo $_POST['name'];?>" />
<input type=hidden name="fname" value="<?php echo $_POST['fname'];?>" />
<input type=hidden name="mname" value="<?php echo $_POST['mname'];?>" />
<input type=hidden name="gen" value="<?php echo $_POST['gen'];?>" />
<tr> <td><input type=reset value=clear></td> <td><input type="submit" value="NEXT>>"></td>
</tr>
</table>
</form>
</body>
</html>
```

Fig: Output of page2.php

page3.php

```
<html>
<head>
<title>Educational Details</title>
</head>
<body>
<form method=POST action="page4.php">
<table align=center>
<tr> <td>SSC Percentage</td><td><input type="text" name="ssc"></td> </tr>
<tr> <td>Intermediate/10+2</td><td><input type="text" name="inter"></td> </tr>
<tr> <td>UG </td><td><input type=text name=ug></td> </tr>
<input type=hidden name="name" value="<?php echo $_POST['name'];?>" />
<input type=hidden name="fname" value="<?php echo $_POST['fname'];?>" />
<input type=hidden name="mname" value="<?php echo $_POST['mname'];?>" />
<input type=hidden name="gen" value="<?php echo $_POST['gen'];?>" />
<input type=hidden name="email" value="<?php echo $_POST['email'];?>" />
<input type=hidden name="Mobile" value="<?php echo $_POST['Mobile'];?>" />
<input type=hidden name="address" value="<?php echo $_POST['address'];?>" />
<tr> <td><input type=reset value=clear></td> <td><input type="submit" value="NEXT>>"></td>
</tr>
</table>
</form>
</body>
</html>
```

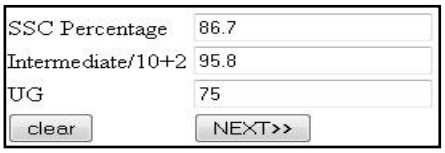


Fig: Output of page3.php

page4.php

```
<html>
<head>
<title>Complete Information</title>
</head>
<body>
<table align=center width=40%>
<tr> <td colspan=2><h5>Personal Details</h5></td> </tr>
<tr> <td>NAME</td><td><?php echo $_POST["name"];?></td> </tr>
<tr> <td>FATHER NAME</td><td><?php echo $_POST["fname"];?></td> </tr>
<tr> <td>MOTHER NAME</td><td><?php echo $_POST["mname"];?></td> </tr>
<tr> <td>GENDER</td><td><?php echo $_POST["gen"];?></td> </tr>
<tr> <td colspan=2><h5>Contact details</h5></td> </tr>
<tr> <td>E-Mail</td><td><?php echo $_POST["email"];?></td> </tr>
<tr> <td>Mobile</td><td><?php echo $_POST["Mobile"];?></td> </tr>
<tr> <td>ADDRESS</td><td><?php echo $_POST["address"];?></td> </tr>
<tr> <td colspan=2><h5>Educational details</h5></td> </tr>
<tr> <td>SSC Percentage</td><td><?php echo $_POST["ssc"];?></td> </tr>
<tr> <td>Intermediate/10+2</td><td><?php echo $_POST["inter"];?></td> </tr> <tr>
<td>UG </td><td><?php echo $_POST["ug"];?></td> </tr>
</table>
</body>
</html>
```

Personal Details	
NAME	suresh
FATHER NAME	sreeramulu
MOTHER NAME	savithri
GENDER	MALE
Contact details	
E-Mail	suresh.csemits@gmail.com
Mobile	8974654610
ADDRESS	sai nagar, tirupati
Educational details	
SSC Percentage	86.7
Intermediate/10+2	95.8
UG	75

Fig: Output of page4.php

As you can see, by passing the values in the hidden form fields, you can continue to collect information.

5 . <u>Redisplaying Forms with Preserved Information and Error Messages</u>

When receiving information submitted from a user, the information may not be submitted in the format you need. To ensure that users do not GET frustrated, it is important to inform them of what they did wrong and clearly tell them how to fix the problem. It is also bad practice to force users to completely rewrite all the proper information they may have already submitted on the form. If users

are forced to do redundant work, they may become irritated and potentially disregard your service altogether. Therefore, to keep users happy, it is important to validate properly and clearly while keeping matters as simple for them as possible.

Example:

```
<html>
<head>
<title> Validation DEMO </title>
<?php
if($_SERVER["REQUEST_METHOD"]=="POST"){
if($_POST["uname"]==—‖){
echo "<font color=red>Please Enter valid User name</font><br>";
}
else if(strlen($_POST["uname"])<6){
echo "<font color=red>Please Enter valid User name with more than 6 characters<br></font>";
}
if($_POST["pwd"]==—‖){
echo "<font color=red>Please Enter valid Password</font><br>";
}
else if(strlen($_POST["pwd"])<6){
echo "<font color=red>Please Enter valid Password with more than 6 characters<br></font>";
}
}
?>
</head>
<body>
<form method=POST action="<?php $_SERVER['PHP_SELF']?>" >
<table>
<tr><td>NAME:</td> <td><input type=text name="uname" /></td> </tr>
<tr><td>PASSWORD:</td> <td><input type=password name="pwd" /></td> </tr>
<tr><td><input type=reset value=CLEAR /></td> <td><input type=submit value=NEXT /></td>
</tr>
</table>
</form> <?php
if($_SERVER["REQUEST_METHOD"]=="POST"){ if($_POST["uname"]!=—‖ &&
strlen($_POST["uname"])>=6 && $_POST["pwd"]!=—‖
&& strlen($_POST["pwd"])>=6){ echo "Name:<font
color=green>".$_POST['uname']."<br></font>"; echo "Password:
<font color=green>".$_POST['pwd']."<br></font>";
}
}
?>
```

</body>
</html>

Fig: Name with less than 6 characters and with out password ***Fig: With out password and with name***

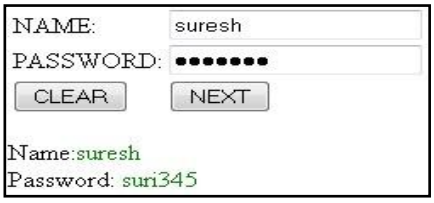


Fig: With valid inputs ..

6. <u>Global & Super global variables</u>

GLOBAL

$GLOBALS References all variables available in global scope. It is an associative array containing references to all variables which are currently defined in the global scope of the script. The variable names are the keys of the array.

Example: <?php
function test() { $foo =
"local content";
echo '$foo in global scope: ' . $GLOBALS["foo"]; echo
'
$foo in current scope: ' . $foo . "\n";
}
$foo = "Global content"; test();
?>

OUTPUT:

$foo in global scope: Global content
$foo in current scope: local content

Superglobal variables

Superglobals are built-in variables that are always available in all scope. Several predefined variables in PHP are "superglobals", which means they are available in all scopes throughout a script. Superglobals were introduced to PHP 4.1. There is no need to do global $variable; to access them within functions or methods. These superglobal variables are: The **$_SERVER** superglobal contains information created by the web serverdetails regarding the server and client configuration and the current request environment.

Examples:

- **$_SERVER['HTTP_REFERER']**: The URL of the page that referred the user to the current location.
- **$_SERVER['REMOTE_ADDR']**: The clients IP address.

- The **$_GET** superglobal contains information pertinent to any parameters passed using the GET method.
- The **$_POST** superglobal contains information pertinent to any parameters passed using the POST method.
- The **$_COOKIE** superglobal stores information passed into the script through HTTP cookies. Such cookies are typically set by a previously executed PHP script through the PHP function **setcookie()**.

Example:

```
<html>
<head>
<title> Validation DEMO </title>
<?php
if($_SERVER["REQUEST_METHOD"]=="POST"){
if($_POST["uname"]==—‖){
echo "<font color=red>Please Enter valid User name</font><br>";
}
else if(strlen($_POST["uname"])<6){
echo "<font color=red>Please Enter valid User name with more than 6 characters<br></font>";
}
if($_POST["pwd"]==—‖){
echo "<font color=red>Please Enter valid Password</font><br>";
}
else if(strlen($_POST["pwd"])<6){
echo "<font color=red>Please Enter valid Password with more than 6 characters<br></font>";
}
}
?>
</head>
<body>
<form method=POST action="<?php $_SERVER['PHP_SELF']?>" > <table>
<tr> <td>NAME:</td> <td><input type=text name="uname" /></td> </tr>
<tr> <td>PASSWORD:</td> <td><input type=password name="pwd" /></td> </tr>
<tr><td><input type=reset value=CLEAR /></td> <td><input type=submit value=NEXT
/></td></tr>
</table>
</form> <?php
if($_SERVER["REQUEST_METHOD"]=="POST"){ if($_POST["uname"]!=—‖ &&
strlen($_POST["uname"])>=6 && $_POST["pwd"]!=—‖
&& strlen($_POST["pwd"])>=6){ echo "Name:<font
color=green>".$_POST['uname']."<br></font>"; echo "Password:
<font color=green>".$_POST['pwd']."<br></font>";
}
}
```

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

7. Preventing Multiple Submissions of a Form

One possible occurrence that happens often is that users become impatient when waiting for your script to do what it is doing, and hence they click the submit button on a form repeatedly. This can create confusion to your script because, while the user may not see anything happening, your script is probably going ahead with whatever it has been programmed to do.

Of particular danger are credit card number submittals. If a user continually hits the submit button on a credit card submittal form, their card may be charged multiple times if the developer has not taken the time to validate against such an eventuality. You can deal with multiple submittal validation in essentially two ways.

- **Server side** refers to a script located on the server that is receiving the data.
- **Client side** is mostly browser related.

7.1 Preventing Multiple Submissions on the Server Side

While you can accomplish this goal in a number of ways from a server-side perspective, we prefer to use a session-based method. Basically, once the submit button has been clicked; the server logs the request from the individual user. If the user attempts to resubmit a request, the script notes a request is already in motion from this user and denies the subsequent request. Once the script has finished processing, the session is unset, and you have no more worries. The following script is an example for Preventing Multiple Submissions on the Server Side

Filename: p1.php

```
<html>
<body>
<form name="test" method="post" action="p2.php">
Name::<input type="text" name="uname"><br>
Password::<input type="password" name="pwd"><br>
<input type="submit" value="SUBMIT" id="submitbut"><br>
</form>
</body>
</html>
```

Filename: p2.php

```
<?php
$name=$_POST['uname'];
$pwd=$_POST['pwd'];
session_start(); if(!isset($_SESSION['x'])){
$_SESSION['x']=TRUE;
}
if($_SESSION['x']==TRUE){ mysql_connect("localhost","root","");
mysql_select_db("TEST");
mysql_query("INSERT INTO login(uname,pwd) VALUES('$name','$pwd')");
$_SESSION['x']=FALSE;
for($i=0;$i<=2000000;$i++);//do nothing for($i=0;$i<=2000000;$i++);//do
nothing for($i=0;$i<=2000000;$i++);//do nothing
```

```
for($i=0;$i<=2000000;$i++);//do nothing for($i=0;$i<=2000000;$i++);//do
nothing
}
echo "Successfully added to database"; session_unset();
?>
```

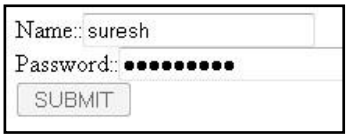
7.2 Preventing Multiple Submissions on the Client Side

Handling multiple submittals from a client-side perspective is actually much simpler than doing it on the server side. With well-placed JavaScript, you can ensure that the browser will not let the submittal go through more than once. The problem with this method, of course, is that JavaScript is not always foolproof because of the user‘s ability to turn it off. The following example uses JavaScript to cut off multiple submittals from a client-side (browser) level.

Filename: p1.php

```
<html>
<head> <script> function
checkandsubmit() {
//Disable the submit button.
document.test.submitbut.disabled = true;
//Then submit the form.
document.test.submit();
}
</script>
</head>
<body>
<form name="test" onsubmit="return checkandsubmit ()" method="post" action="p2.php">
Name::<input type="text" name="uname"><br>
Password::<input type="password" name="pwd"><br>
<input type="submit" value="SUBMIT" id="submitbut"><br>
</form>
</body>
</html>
```

After submitting, the button will be disabled as follows



8. Handling Special Characters

An added security feature, particularly when dealing with database submittal, is validating against special characters being inserted into your script. Be it a database insertion script, a contact form, or even a mailer system, you always want to ensure that no malicious users are attempting to sabotage your script with bad (or special) characters. PHP allots a number of functions to use in this regard.

```
string trim ( string str [, string charlist] )
string htmlspecialchars ( string string [, int quote_style [, string charset]] )
string strip_tags ( string str [, string allowable_tags] ) string addslashes (
string str )
```

The following script demonstrates above functions

```
<?php
$msg1=" Welcome to PHP ";//for trim
$msg2="<b>Welcome to php</b>"; $msg3="Welcome \n
to \php"; echo "With out using trim():
**".$msg1."**<br>"; echo "Using
trim():**".trim($msg1)."**<br><br><br>"; echo "With
out using htmlspecialchars(): ".$msg2."<br>";
echo "Using htmlspecialchars(): ".htmlspecialchars($msg2)."<br><br><br>";
echo "Using strip_tags(): ".strip_tags($msg2)."<br><br><br>"; echo "With
out Using addslashes(): ".$msg3."<br>";
echo "Using addslashes(): ".addslashes($msg3)."<br><br><br>"; ?>
```

OUTPUT:

With out using trim(): ** Welcome to PHP **
Using trim():**Welcome to PHP**
With out using htmlspecialchars(): **Welcome to php**
Using htmlspecialchars(): Welcome to php
Using strip_tags(): Welcome to php
With out Using addslashes(): Welcome to \php Using
addslashes(): Welcome to \\php

- The **trim()** function removes any blank space found at the beginning or end of the submitted string.
- The **htmlspecialchars()** function turns attempted HTML into its special character equivalent.
- The **strip_tags()** function completely removes any characters it sees as being a tag.
- **addslashes()**, places a slash in front of any characters that could be harmful to the database such as apostrophes. The end result is a string that is quite squeaky clean.

9. File Uploads

Handling file uploads in PHP is not exactly difficult from a syntax point of view, but it is important (extremely important in fact) to ensure that the file being uploaded is within the upload constraints you lay out for it. The following are the constrains of File:

- **size:** The size of the uploaded file (in bytes). You could easily find your server under some heavy loads if you are not careful about what size of files are being uploaded.
- **type:** The MIME type of the uploaded file. Ex: .jpeg, .pdf, .doc, etc,.
- **name:** The original file name that was uploaded. It is possible to change file name at the time of uploading.
- **tmp_name:** The temporary name of the file that has been uploaded.
- **error:** The error code that may be generated by the file upload.

The following Example demonstrate file uploading using Form <form
action="" method="post" enctype="multipart/form-data">
<p>Upload Pictures:
<input type="file" name="pictures[]" />
<input type="file" name="pictures[]" />
<input type="file" name="pictures[]" />
<input type="submit" value="Send" name="submitbut"/>

```
</p>
</form> <?php
if(isset($_POST['submitbut']))
{
foreach ($_FILES["pictures"]["error"] as $key => $error) { if
($error == UPLOAD_ERR_OK) {
$tmp_name = $_FILES["pictures"]["tmp_name"][$key]; $name
= $_FILES["pictures"]["name"][$key];
move_uploaded_file($tmp_name, "data/$name");
}
}
}
?>
```

11 . Questions

1. (a) List and describe the different form elements associated with common form issues.

(b) Differentiate GET and POST methods. *[Reg-April/May 2012(Set-1) IT, Reg-Nov 2012 (set-4) CSE]]*

2. Explain with example how the validation of forms is done using PHP. *[Reg-April/May 2012(Set2) IT]*

3. (a) What is the advantage of super globals, explain with example?

(b) Write a PHP program to submit values using super globals and globals. *[RegApril/May 2012(Set-3) IT]*

4. How can we prevent multiple submissions of a form on server side? Explain with example. *[RegApril/May 2012(Set-4) IT]*

[Reg-January 2014 (set-3) CSE]

5. (a) What is the advantage of Superglobals? Explain with example.

(b) Write a program to differentiate GET and POST methods. *[Reg-Nov 2012 (set-1)]* 6. (a) Explain briefly about the POST method with example.

(b) Differentiate Superglobals versus Globals. *[Reg-Nov 2012 (set-2)]* 7.

(a) Explain briefly about the GET method with example.

(b) Differentiate GET and POST methods. *[Reg-Nov 2012 (set-3)]*

8. (a) What are the disadvantages of redisplaying forms without previous information and error messages?

(b) Write a PHP program for redisplaying forms with previous information and error messages. *[sup-Nov2012]*

9. (a) What are the disadvantages of redisplaying forms without previous information and error messages?

(b) Explain briefly the different constrains in uploading a file. *[Reg-January 2014 (set-4) CSE]*

10. (a) Explain briefly about the GET and POST methods.

(b) Write a program to differentiate GET and POST methods. *[Reg-January 2014 (set-2) CSE]*

11. Explain the following functions handling special characters: (a) trim()

(b) htmlspecialchars()

(c) strip_tags()

(d) addslashes() *[Reg-January 2014 (set-1) CSE]*

1.XML:

What is XML?

Extensible Markup Language (XML) is used to describe data. The XML standard is a flexible way to create information formats and electronically share structured data via the public Internet, as well as via corporate networks.

Short for ***Ex**tensible **M**arkup **L**anguage*, a specification developed by the W3C. XML is a pareddown version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

- XML was designed to **describe data**.
- XML tags are not predefined in XML. You must **define your own tags**.
- XML is **self describing**.
- XML uses a DTD (**Document Type Definition**) to formally describe the data.

2. **XML BASICS**

XML, or eXtensible markup language, is all about creating a universal way for both formatting and presenting data. Once data is coded or marked up with XML tags, data can then be used in many different ways.

Main features of XML:

- XML files are text files, which can be managed by any text editor. ➢ XML is very simple, because it has less than 10 syntax rules.
- Because of these features, XML offers following **advantages**
- XML provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations. XML data is stored in plain text format.
- 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.
- Databases can trade tables, business applications can trade updates, and document systems can share information.
- It supports Unicode, allowing almost any information in any written human language to be communicated.
- Its self-documenting format describes structure and field names as well as specific values.
- Content-based XML markup enhances searchability, making it possible for agents and search engines to categorize data instead of wasting processing power on context-based fulltext searches.
- XML is heavily used as a format for document storage and processing, both online and offline.

- It is based on international standards.
- It is platform-independent, thus relatively immune to changes in technology.
- Forward and backward compatibility are relatively easy to maintain despite changes in DTD or Schema.

How can XML be used?

- XML can keep data separated from HTML
- XML can be used to store data inside HTML documents

- XML can be used as a format to exchange information
- XML can be used to store data in files or in databases

An example XML document:

<?xml version="1.0"?> // It defines the XML version of the document.
<note> // the first element of the document (the root element):
<to>Abhi</to> // defines 4 child elements of the root (to, from, heading, and body)
<from>Avi</from>
<heading>Reminder</heading>
<body>Please send me the details of SAP!</body>
</note> // the end of the root element

Main points to be considered in XML

1. In XML all elements must have a closing tag like this: This is a paragraph
2. XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.
3. In XML all elements must be properly nested within each other like this:

This text is bold and italic

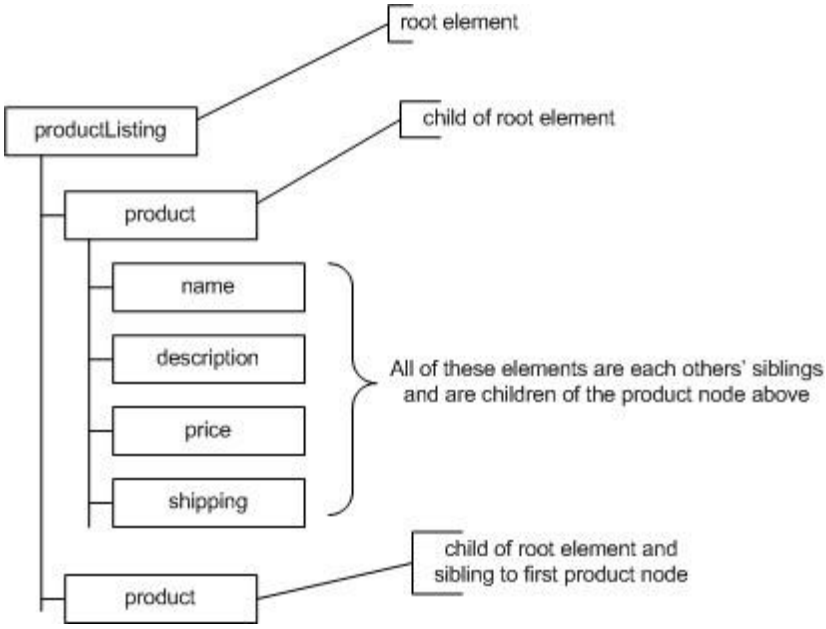
4. All XML documents must contain a single tag pair to define the root element. All other elements must be nested within the root element. All elements can have sub (children) elements. Sub elements must be in pairs and correctly nested within their parent element:

<root>
<child>
<subchild>
</subchild>
</child> </root>

5. XML elements can have attributes in name/value pairs just like in HTML. In XML the attribute value must always be quoted.

<?xml version="1.0"?>
<note date=12/11/99> // Incorrect
<note date="12/11/99"> // Correct
<to>Abhi</to>
<from>Avi</from>
<heading>Reminder</heading>
<body>Don't forget the visit!</body>
</note>

The logical structure of an XML document.



The main difference between XML and HTML

HTML is an abbreviation for HyperText Markup Language while XML stands for eXtensible Markup Language.The *differences* are as follows:-

1. HTML was designed **to display data with focus on how data looks** while XML was designed to be a software and hardware independent tool used to **transport and store data, with focus on what data is**.
2. HTML is a **markup language** itself while XML provides **a framework for defining markup languages**.
3. HTML is a **presentation language** while XML is **neither a programming language nor a presentation language**.
4. HTML is **case insensitive** while XML is **case sensitive**.
5. HTML is **used for designing a web-page** to be rendered on the client side while XML is used basically to **transport data** between the application and the database.
6. HTML has its **own predefined tags** while what makes XML flexible is that **custom tags** can be defined and the tags are invented by the author of the XML document.
7. HTML is **not strict** if the user does not use the **closing tags** but XML makes it **mandatory** for the user the close each tag that has been used.
8. HTML does **not preserve white space** while XML **does**.

9.HTML is about displaying data, hence **static** but XML is about carrying information, hence **dynamic**. Thus, it can be said that HTML and XML are not competitors but rather **complement** to each other and clearly serving altogether different purposes.

Types of XML Documents

There are two kinds of XML documents:

1. Well-formed

A "Well Formed" XML document is a document that conforms to the XML syntax rules. They contain text and XML tags. Everything is entered correctly. They do not, however, refer to a DTD. The following is a "Well Formed" XML document:

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

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

2. Valid

Valid documents not only conform to XML syntax but they also are error checked against a **Document Type Definition** (DTD) or schema

The following is the same document as above but with an added reference to a DTD:

<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "InternalNote.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>

The building blocks of XML documents

XML documents (and HTML documents) are made up by the following building blocks:- Elements, Tags, Attributes, Entities, PCDATA, and CDATA This is a brief explanation of each of the building blocks:

Elements

Elements are the main building blocks of both XML and HTML documents. Examples of HTML elements are "body" and "table".

Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img". In a DTD, elements are declared with an ELEMENT declaration.

Tags

Tags are used to markup elements.

A starting tag like <element_name> mark up the beginning of an element, and an ending tag like </element_name> mark up the end of an element.

Examples: A body element: <body>body text in between</body>. A message element: <message>some message in between</message>

Attributes

Attributes provide extra information about elements.

Attributes are placed inside the start tag of an element. Attributes come in name/value pairs. The following "img" element has an additional information about a source file:

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

PCDATA

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.

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. **Entities**

Entities as variables used to define common text. Entity references are references to entities. Most of you will known the HTML entity reference: " " that is used to insert an extra space in an HTML document. Entities are expanded when a document is parsed by an XML parser. The following entities are predefined in XML:

: **Entity References**	**Character**
<	<
>	>
&	&
"	"
'	'

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

3. Document Type Definition

Introduction to DTD

The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in your XML document, or as an external reference.

Internal DTD

This is an XML document with a Document Type 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 is interpreted like this:

!ELEMENT note (in line 2) defines the element "note" as having four elements:

"to,from,heading,body". **!ELEMENT to** (in line 3) defines the "to" element to be of the type "CDATA". **!ELEMENT from** (in line 4) defines the "from" element to be of the type "CDATA" and so on.....

External DTD

This is the same XML document with an external DTD:

```
<?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>
```

This is a copy of the file "note.dtd" containing the Document Type Definition:

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

Example(Program) on DTD

This is an example of an XML document that used an external DTD file and cascading style sheet (CSS) file.

The XML File

The following file is called "parts.xml".

```
<?xml version="1.0"?>
<!DOCTYPE PARTS SYSTEM "parts.dtd">
<?xml-stylesheet type="text/css" href="xmlpartsstyle.css"?>
<PARTS>
<TITLE>Computer Parts</TITLE>
<PART>
<ITEM>Motherboard</ITEM>
<MANUFACTURER>ASUS</MANUFACTURER>
<MODEL>P3B-F</MODEL>
<COST> 123.00</COST>
</PART>
<PART>
<ITEM>Video Card</ITEM>
<MANUFACTURER>ATI</MANUFACTURER>
<MODEL>All-in-Wonder Pro</MODEL>
<COST> 160.00</COST>
</PART>
```

<PART>
<ITEM>Sound Card</ITEM>
<MANUFACTURER>Creative Labs</MANUFACTURER>
<MODEL>Sound Blaster Live</MODEL>
<COST> 80.00</COST>
</PART>
<PART>
<ITEM>9 inch Monitor</ITEM>
<MANUFACTURER>LG Electronics</MANUFACTURER>
<MODEL> 995E</MODEL>
<COST> 290.00</COST>
</PART>
</PARTS>
This file specifies the use of two external files.
 A DTD file called "parts.dtd". This is done on the second line:
<!DOCTYPE PARTS SYSTEM "parts.dtd">

The Style File
The following file is used to set the style of the elements in the XML file. It is called "xmlpartstyle.css".
PARTS
{ display: block }
TITLE
{ display: block; font-
family: arial; color:
#008000; font-weight:
600; font-size: 22;
margin-top: 12pt;
text-align: center }
PART
{ display: block }
ITEM
{ display: block; font-
family: arial; color:
#000080; font-weight:
400; margin-left: 15pt;
margin-top: 12pt;
font-size: 18 }

MANUFACTURER
{ display: block; font-family: arial; color: #600060; font-weight: 400; margin-left: 45pt; margin-top: 5pt; font-size: 18 }
MODEL
{ display: block; font-family: arial; color: #006000; font-weight: 400; margin-left: 45pt; margin-top: 5pt; font-size: 18 }
COST
{ display: block; font-family: arial; color: #800000; font-weight: 400; margin-left: 45pt; margin-top: 5pt; font-size: 18 }

Why use a DTD?

XML provides an application independent way of sharing data. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. Your application can use a standard DTD to verify that data that you receive from the outside world is valid. You can also use a DTD to verify your own data.

A lot of forums are emerging to define standard DTDs for almost everything in the areas of data exchange.

4. XML Schema

A *schema* formally describes what a given XML document contains, in the same way a database schema describes the data that can be contained in a database (table structure, data types). An XML schema describes the coarse shape of the XML document, what fields an element can contain, which sub elements it can contain, and so forth. It also can describe the values that can be placed into any element or attribute.

An **XML Schema** is a language for expressing constraints about **XML** documents. There are several different **schema** languages in widespread use, but the main ones are Document Type Definitions (DTDs), Relax-NG, Schematron and W3C **XSD** (**XML Schema** Definitions).

What is XML Schema Used For?

A Schema can be used:
- to provide a list of elements and attributes in a vocabulary;
- to associate types, such as integer, string, etc., or more specifically such as hatsize, sock_colour, etc., with values found in documents;

- to constrain where elements and attributes can appear, and what can appear inside those elements, such as saying that a chapter title occurs inside a chapter, and that a chapter must consist of a chapter title followed by one or more paragraphs of text;
- to provide documentation that is both human-readable and machine-processable; ○ to give a formal description of one or more documents.

What is the difference between XML Schema and DTD? i)
DTD is the predecessor of XML schema.
ii) While DTD provides the basic structure/grammar for defining a XML document, in addition to that XML schema provides methods to define constraints on the data contained in the document. Therefore XML schema is considered to be richer and powerful than DTD. iii) Also, XML schema provides an object oriented approach for defining the structure of a XML document. But since XML schema is a new technology, some XML parsers do not support it yet. iv) XML Schema is namespace aware, while DTD is not.
v) XML Schemas are written in XML, while DTDs are not.
vi) XML Schema is strongly typed, while DTD is not.
vii) XML Schema has a wealth of derived and built-in data types that are not available in DTD.
viii) XML Schema does not allow inline definitions, while DTD does.

DTD vs. XSD **DTD**	**XSD**
DTD stands for **Document Type Definition**.	XSD stands for XML Schema Definition.
DTDs are derived from **SGML** syntax.	XSDs are written in XML.
DTD doesn't support datatypes.	XSD supports datatypes for elements and attributes.
DTD doesn't support namespace.	XSD supports namespace.
DTD doesn't define order for child DTD is not extensible.	XSD defines order for child elements. elements. XSD is extensible.
DTD provides less control on XML structure.	XSD provides more control on XML structure.

Example(Program) on Schema
This example attempts to validate an XML data file (books.xml) against an XML schema definition file (books.xsd). According to books.xsd, each <book> element must have a <pub_date> child element. The second <book> element in books.xml does not have this required element. Therefore, when we attempt to validate the XML file , we should get a validation error.

XML file(books.xml)

```
<?xml version="1.0"?>
<x:books xmlns:x="urn:books">
<book id="bk001">
<author>Writer</author>
<title>The First Book</title>
```

<genre>Fiction</genre>
<price>44.95</price>
<pub_date>2000-10-01</pub_date>
<review>An amazing story of nothing.</review>
</book>
<book id="bk002">
<author>Poet</author>
<title>The Poet's First Poem</title>
<genre>Poem</genre>
<price>24.95</price>
<review>Least poetic poems.</review>
</book>
</x:books>

5. DOM and Presenting XML Document Object Model

XML parsers can handle documents in any way that their developers choose. There are two models commonly used for parsers i.e., SAX and DOM. SAX parsers are used when dealing with streams of data. This type of parsers is usually used with java. SAX-based parsers run quickly.

DOM is and application program interface (API) for XML documents. The DOM API specifies the logical structure of XML documents and the ways in which they can be accessed and manipulated. The DOM API is just a specification. DOM-complaint applications include all of the functionality needed to handle XML documents. They can build static documents, navigate and search through them, add new elements, delete elements, and modify the content of existing elements. The views XML document as trees. The DOM exposes the whole of the document to applications. It is also scriptable so applications can manipulate the individual nodes.

Presenting XML

XML documents are presented using Extensible Stylesheet which expresses stylesheets. XSL stylesheet are not the same as HTML cascading stylesheets. They create a style for a specific XML element, with XSL a template is created. XSL basically transforms one data structure to another i.e., XML to HTML.

6.XML Parsers and Validation

XMl Parsers

Parsing XML refers to going through XML document to access data or to modify data in one or other way. Parser has the job of reading the XML, checking it for errors, and passing it on to the intended application. If no DTD or schema is provided, the parser simply checks that the XML is well-formed. If a DTD is provided then the parser also determines whether the XML is valid, i.e. that the tags, attributes, and content meet the specifications found in the DTD, before passing it on to the application.

Why do we need XML Parsers

We need XML parser because we do not want to do everything in our application from scratch, and we need some "helper" programs or libraries to do something very low-level but very necessary to us. ○ These low-level but necessary things include checking the well-formedness,

validating the document against its DTD or schema (just for validating parsers), resolving character reference, understanding CDATA sections, and so on.

- XML parsers are just such "helper" programs and they will do all these jobs.

Types of parsers: SAX and DOM

Simple API for XML (SAX)

Simple API for XML (SAX) parsing is different from DOM as it parses the XML files step by step and is event based model. The SAX parser triggers an event when they encounter an opening tag, element or attribute. Unlike in DOM parser it is advisable to use the SAX parser for parsing large XML documents as it does not load the complete XML file in the memory. This parser parses node by node so it can read large XML files in smaller parts.

Difference between SAX and DOM **DOM**	 **SAX**
Tree model parser (Tree of nodes)	Event based parser (Sequence of events)
DOM loads the file into the memory and then parse the file	SAX parses the file at it reads i.e. Parses node by node
Has memory constraints since it loads the whole XML file before parsing	No memory constraints as it does not store the XML content in the memory
DOM is read and write (can insert or delete the node)	SAX is read only i.e. can‘t insert or delete the node
If the XML content is small then prefer DOM parser	Use SAX parser when memory content is large
Backward and forward search is possible for searching the tags and evaluation of the inside the tags. So this gives the ease of navigation	SAX reads the XML file from top to bottom and backward navigation is not possible information
Slower at runtime	Faster at runtime

That‘s all on difference between SAX and DOM parsers in Java, now it‘s up to you on which XML parser you going to choose. DOM parser is recommended over SAX parser if XML file is small enough and go with SAX parser if you don‘t know size of xml files to be processed or they are large.

What are the usual application for a DOM parser and for a SAX parser?

In the following cases, using SAX parser is advantageous than using DOM parser.

- The input document is too big for available memory
- You can process the document in small contiguous chunks of input. You do not need the entire document before you can do useful work
- You just want to use the parser to extract the information of interest, and all

In the following cases, using DOM parser is advantageous than using SAX parser.

- Your application needs to access widely separately parts of the document at the same time.
- Your application may probably use an internal data structure which is almost as complicated as the document itself.

- Your application has to modify the document repeatedly. ○ Your application has to store the document for a significant amount of time through many method calls.

Sample document for the example

```
<?xml version="1.0"?>
<!DOCTYPE shapes [
<!ELEMENT shapes (circle)*>
<!ELEMENT circle (x,y,radius)>
<!ELEMENT x (#PCDATA)>
<!ELEMENT y (#PCDATA)>
<!ELEMENT radius (#PCDATA)>
<!ATTLIST circle color CDATA #IMPLIED>
]>
<shapes>
<circle color="BLUE">
<x>20</x>
<y>20</y>
<radius>20</radius>
</circle>
<circle color="RED" >
<x>40</x>
<y>40</y>
<radius>20</radius>
</circle>
</shapes>
```

XML Validation

XML file can be validated by 2 ways:

1. against DTD
2. against XSD

DTD (Document Type Definition) and XSD (XML Schema Definition) are used to define XML structure.

XML DTD

In our XML tutorial, you will learn about DTD file, creating xml with DTD, using CSS file, CDATA vs PCDATA and difference between DTD and XML schema.

Let's see an example of XML using DTD file. *employee.xml*

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

</employee>

Test it Now

1)	DTD	Document type definition	It is an standard which is used to define the legal elements in an XML document.
2)	XSD	XML schema definition	It is an XML based alternative to dtd. It is used to describe the structure of an XML document.

DTD vs XSD

There are many differences between DTD (Document Type Definition) and XSD (XML Schema Definition). In short, DTD provides less control on XML structure whereas XSD (XML schema) provides more control.

The important differences are given below:

S.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.

7.XSL and XSLT Transformation:

XSLT Introduction

XSL (eXtensible Stylesheet Language) is a styling language for XML.
XSLT stands for XSL Transformations.
This tutorial will teach you how to use XSLT to transform XML documents into other formats (like transforming XML into HTML).
Online XSLT Editor
With our online editor, you can edit XML and XSLT code, and click on a button to view the result.
XSLT Example

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

What You Should Already Know
Before you continue you should have a basic understanding of the following:

- HTML
- XML

If you want to study these subjects first, find the tutorials on our Home page.
XSLT References
XSLT Elements
Description of all the XSLT elements from the W3C Recommendation, and information about browser support.
XSLT, XPath, and XQuery Functions

XSLT 2.0, XPath 2.0, and XQuery 1.0, share the same functions library. There are over 100 built-in functions. There are functions for string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, and more.

XSL(T) Languages

XSLT is a language for transforming XML documents.

XPath is a language for navigating in XML documents.

XQuery is a language for querying XML documents.

It Started with XSL

XSL stands for E**X**tensible **S**tylesheet **L**anguage.

The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language.

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.

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

With the **CSS3 Paged Media Module**, W3C has delivered a new standard for document formatting. So, since 2013, CSS3 is proposed as an XSL-FO replacement.

What is XSLT?

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

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.

If you want to study XPath first, please read our XPath Tutorial.

How Does it Work?

In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

XSLT Browser Support

All major browsers support XSLT and XPath.

XSLT is a W3C Recommendation

XSLT became a W3C Recommendation 16. November 1999.

XSLT - Transformation

Example study: How to transform XML into XHTML using XSLT?

The details of this example will be explained in the next chapter.

Correct Style Sheet Declaration

The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.

Note: <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> or:
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.

The xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute version="1.0".

Start with a Raw XML Document

We want to **transform** the following XML document ("cdcatalog.xml") into XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
```

```
  </cd>
.
.
</catalog>
```

Viewing XML Files in IE, Chrome, Firefox, Safari, and Opera: Open the XML file (click on the link below) - The XML document will be displayed with color-coded root and child elements (except in Safari). Often, there is a plus (+) or minus sign (-) to the left of the elements that can be clicked to expand or collapse the element structure. **Tip: To view the raw XML source, rightclick in XML file and select "View Source"!**

Create an XSL Style Sheet

Then you create an XSL Style Sheet ("cdcatalog.xsl") with a transformation template: <?xml version="1.0" encoding="UTF-8"?>

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Link the XSL Style Sheet to the XML Document

Add the XSL style sheet reference to your XML document ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
```

```
  <country>USA</country>
  <company>Columbia</company>
  <price>10.90</price>
  <year>1985</year>
 </cd>
</catalog>
```

If you have an XSLT compliant browser it will nicely **transform** your XML into XHTML.

8. News Feed (RSS and ATOM) ➢

The benefit of RSS and Atom.

- ➢ What is RSS about.
- ➢ Structure and Syntax of RSS.
- ➢ What is Atom about.
- ➢ Structure and Syntax of Atom.
- ➢ What is RSS and Atom all about?

They are a form of communication using XML documents to broadcast information updates to a large group of subscribers.

- RSS stands for Really Simple Syndication. It's an easy way for you to keep up with news and information that's important to you, and helps you avoid the conventional methods of browsing or searching for information on websites. Now the content you want can be delivered directly to you without cluttering your inbox with e-mail messages. This content is called a "feed.—
- RSS is written in the Internet coding language known as XML (eXtensible Markup Language).

➢ RSS must be the root element followed by one channel element.

➢ Required channel elements:

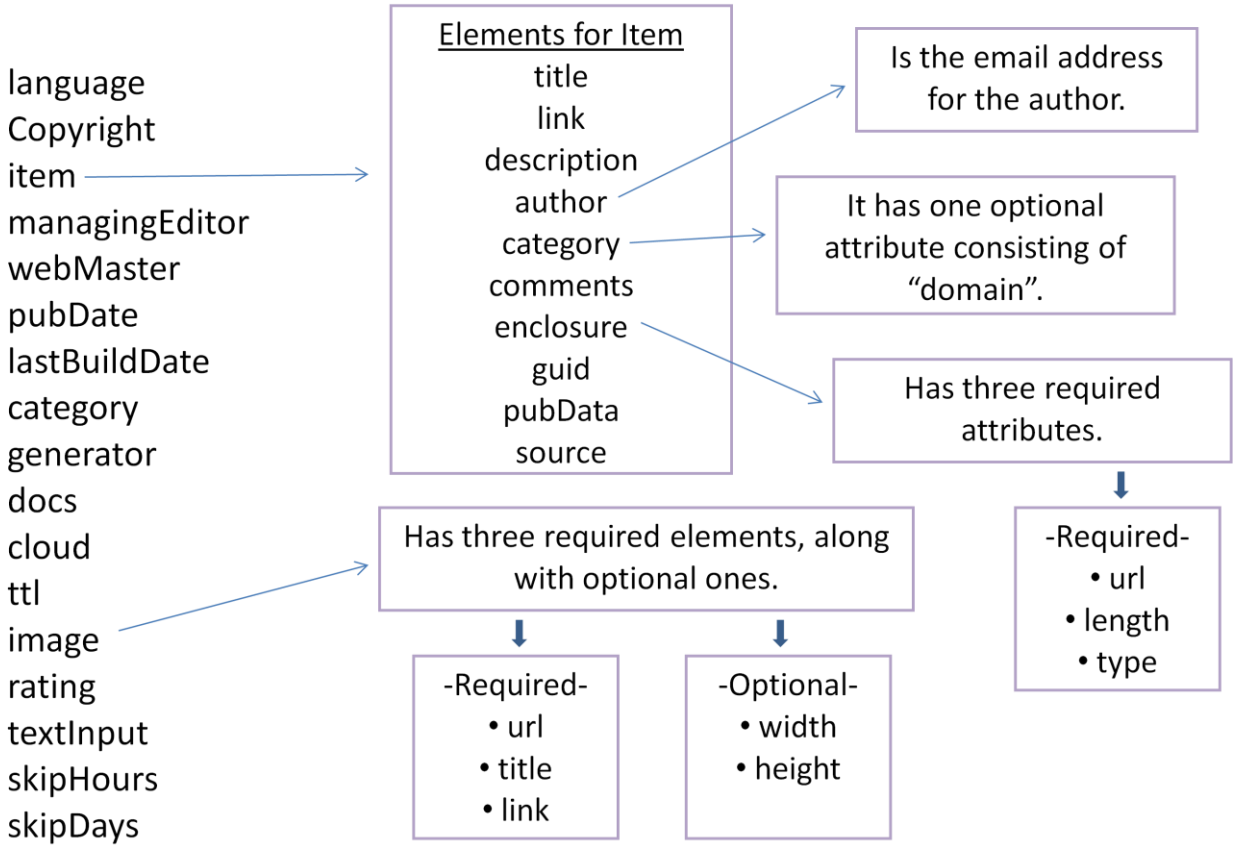
✓ Title: The name of the channel. It's how people refer to your service. If you have an HTML website that contains the same information as your RSS file, the title of your channel should be the same as the title of your website.

✓ Link: The URL to the HTML website corresponding to the channel.

✓ description : Phrase or sentence describing the channel.

RSS 2.0

Optional <channel > Elements




```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Liftoff News</title>
    <link>http://liftoff.msfc.nasa.gov/</link>
    <description>Liftoff to Space Exploration.</description>
    <language>en-us</language>
    <pubDate>Tue, 10 Jun 2003 04:00:00 GMT</pubDate>
    <lastBuildDate>Tue, 10 Jun 2003 09:41:01 GMT</lastBuildDate>
    <docs>http://blogs.law.harvard.edu/tech/rss</docs>
    <generator>Weblog Editor 2.0</generator>
    <managingEditor>editor@example.com</managingEditor>
    <webMaster>webmaster@example.com</webMaster>
    <item>
      <title>Star City</title>
      <link>http://liftoff.msfc.nasa.gov/news/2003/news-starcity.asp</link>
      <description>How do Americans get ready to work with Russians aboard the International
Spac
 Station? They  take a crash course in culture, language and protocol at Russia's &lt;a
href="http://howe.iki.rssi.ru/GCTC/gctc_e.htm"&gt;Star City&lt;/a&gt;.
      </description>
      <pubDate>Tue, 03 Jun 2003 09:39:21 GMT</pubDate>
```

<guid>http://liftoff.msfc.nasa.gov/2003/06/03.html#item573</guid>
<item>
<title>The Engine That Does More</title>
<link>http://liftoff.msfc.nasa.gov/news/2003/news-VASIMR.asp</link>
<description>Before man travels to Mars, NASA hopes to design new engines that will let us fly
through the Solar System more quickly. The proposed VASIMR engine would do that.
</description>
<pubDate>Tue, 27 May 2003 08:37:32 GMT</pubDate>
<guid>http://liftoff.msfc.nasa.gov/2003/05/27.html#item571</guid>
</item>
</channel>
</rss>

What is ATOM 1.0 ?

- ATOM – The Atom Syndication Format is the next generation of XML-based file formats, designed to allow information--the contents of web pages, for example--to be syndicated between applications. Like RSS before it, Atom places the content and metadata of an internet resource into a machine-parsable format, perfect for displaying, filtering, remixing, and archiving.

Atom

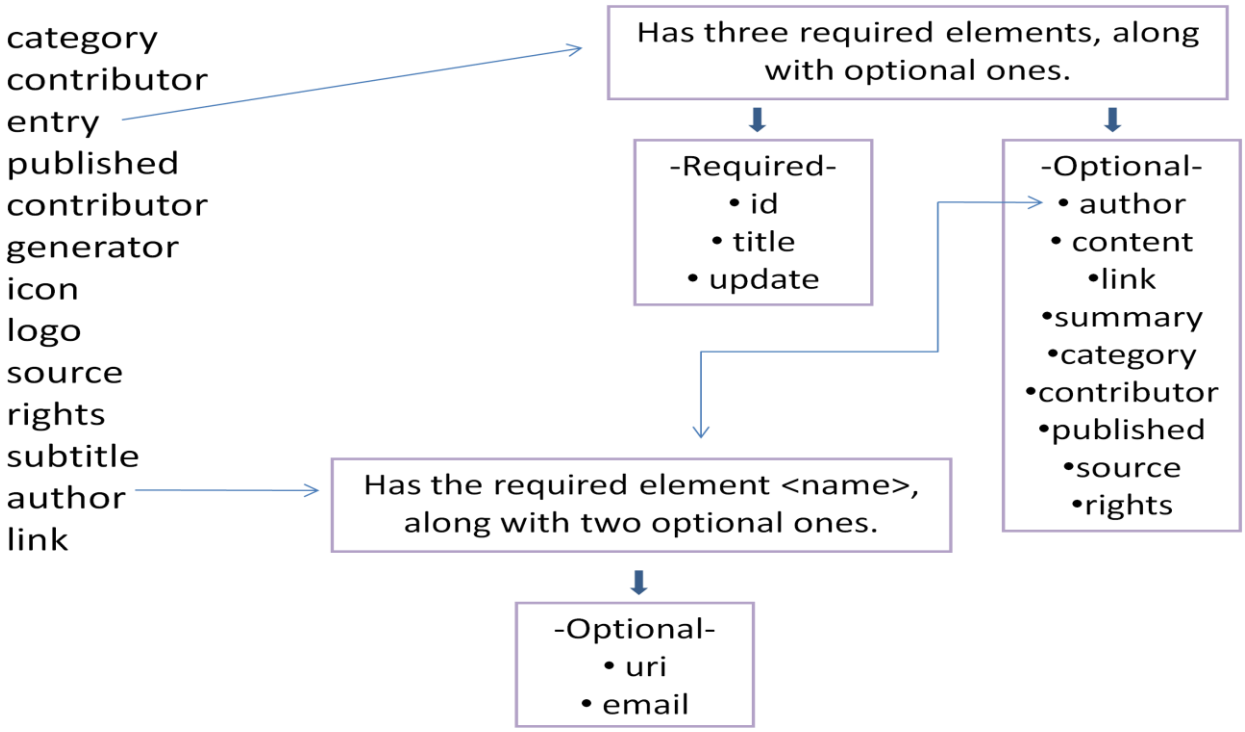
- In Atom, <feed> is used for the root element.

General considerations:
- All elements described in Atom must be in the http://www.w3.org/2005/Atom namespace.
- xml:lang may be used to identify the language of any human readable text.
- xml:base may be used to control how relative URIs are resolved.

- Required <feed> elements:

✓id	A unique identifier, which can be as simple as the URI of a blog entry or other Web resource represented by an entry.
✓title	Contains a human readable title for the feed. Often the same as the title of the associated website.
✓updated	A timestamp which indicates when the last update occurred. (Must conform to RFC 3339)

Atom



Atom Sample

```
<?xml version="1.0"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <link rel="self" href="http://example.org/blog/index.atom"/>
  <id>http://example.org/blog/index.atom</id>
  <icon>../favicon.ico</icon>
  <title>An Atom Sampler</title>
  <subtitle>No Splitting</subtitle>
  <author>
    <name>Ernie Rutherford </name>
    <email>ernie@example.org</email>
    <uri>.</uri>
  </author>
  <updated>2006-10-25T03:38:08-04:00</updated>
  <link href="."/>
  <entry>
    <id>tag:example.org,2004:2417</id>
    <link href="2006/10/23/moonshine"/>
    <title>Moonshine</title>
<content type="text">
      Anyone who expects a source of power from the transformation of the atom is talking moonshine.
    </content>
    <published>2006-10-23T15:33:00-04:00</published>
```

```
    <updated>2006-10-23T15:47:31-04:00</updated>
  </entry>
  <entry>
    <id>>tag:example.org,2004:2416</id>
    <link href="2006/10/21/think"/>
    <title type="html">&lt;strong&gt;Think!&lt;/strong&gt;</title>
    <content type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <p>We haven't got the money, so we've got to think!</p>
      </div>
    </content>
    <updated>2006-10-21T06:02:39-04:00</updated>
  </entry>
</feed>
```

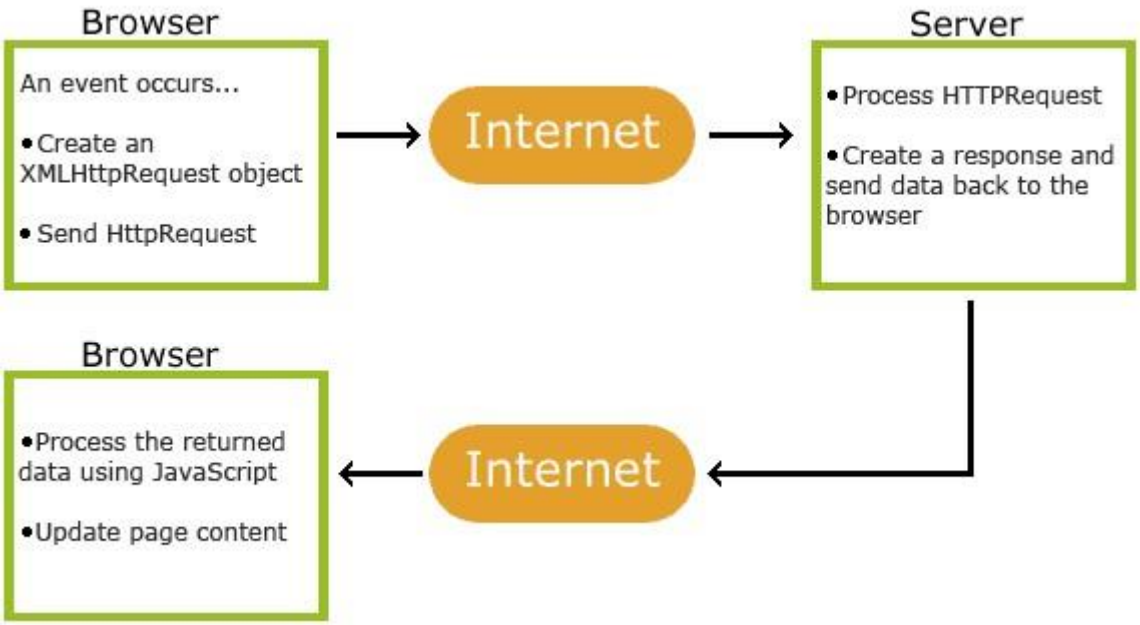
Unit-5

1. AJAX: Ajax Client Server Architecture

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.

How AJAX Works



Example

```
<html>
<body>
<div id="demo">
<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Change Content</button> </div>
<script> function loadDoc() {    var xhttp = new
XMLHttpRequest();   xhttp.onreadystatechange =
function() {     if (this.readyState == 4 &&
this.status == 200) {
     document.getElementById("demo").innerHTML = this.responseText;
   }
  };
  xhttp.open("GET",          "ajax_info.txt",          true);
xhttp.send();
}
</script>
</body>
</html>
```

2. Http Request Object

The XMLHttpRequest object is the key to AJAX. 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.

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").

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>The XMLHttpRequest Object</h1>
<p id="demo">Let AJAX change this text.</p>
```

```
<button type="button" onclick="loadDoc()">Change Content</button>
<script>       function
loadDoc() {
  var xhttp;
  if   (window.XMLHttpRequest)   {
// code for modern browsers
    xhttp = new XMLHttpRequest();
    } else {
    // code for IE6, IE5
    xhttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
  xhttp.onreadystatechange  =  function()  {          if
(this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET",          "ajax_info.txt",          true);
xhttp.send();
}
</script>
</body>
</html>
```

3. Call Back Methods

A callback function is executed after the current effect is 100% finished.

jQuery Callback Functions

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical syntax: **$(*selector*).hide(*speed,callback*);**

Examples

The example below has a callback parameter that is a function that will be executed after the hide effect is completed:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script> <script>
$(document).ready(function(){
   $("button").click(function(){
$("p").hide("slow", function(){
          alert("The paragraph is now hidden");
       });
   });
```

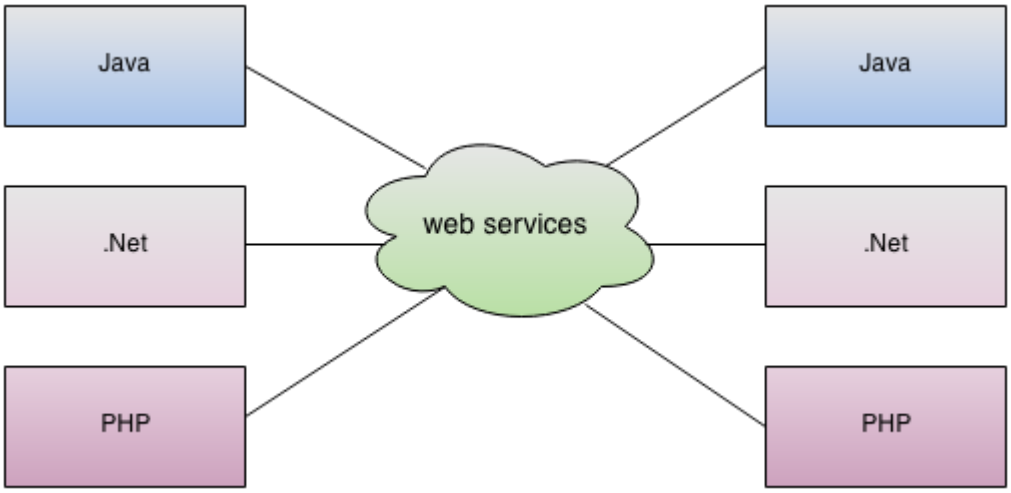
```
});
</script>
</head>
<body>
<button>Hide</button>
<p>This is a paragraph with little content.</p>
</body> </html>
```

Output:

This is a paragraph with little content.

4. Web Services: Introduction

- Web services are open standard (XML, SOAP, HTTP, etc.) based web applications that interact with other web applications for the purpose of exchanging data. Web services can convert your existing applications into web applications.
- Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains.
- Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.
- A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.



Components of Web Services

The basic web services platform is XML + HTTP. All the standard web services work using the following components −

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)

- WSDL (Web Services Description Language) **How Does a Web Service Work?**

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of −

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

Example

Consider a simple account-management and order processing system.

he steps to perform this operation are as follows −

- The client program bundles the account registration information into a SOAP message. This SOAP message is sent to the web service as the body of an HTTP POST request.
- The web service unpacks the SOAP request and converts it into a command that the application can understand.
- The application processes the information as required and responds with a new unique account number for that customer.
- Next, the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
- The client program unpacks the SOAP message to obtain the results of the account registration process.

5. Java web services Basics

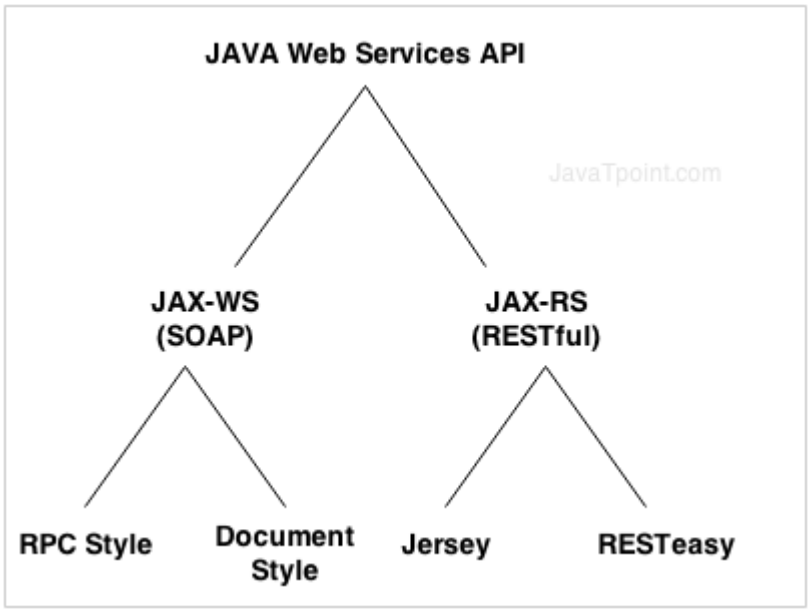
Java web service application perform communication through WSDL (Web Services Description Language). There are two ways to write java web service application code: SOAP and RESTful.

Java Web Services API

There are two main API's defined by Java for developing web service applications since JavaEE 6.

1) **JAX-WS**: for SOAP web services. The are two ways to write JAX-WS application code: by RPC style and Document style.

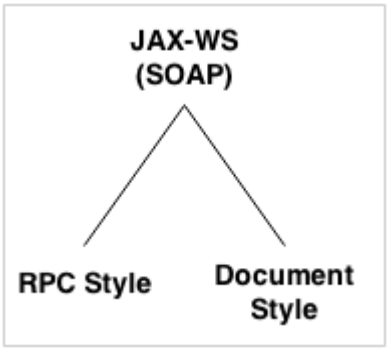
2) **JAX-RS**: for RESTful web services. There are mainly 2 implementation currently in use for creating JAX-RS application: Jersey and RESTeasy.



1) **JAX-WS**:

here are two ways to develop JAX-WS example.

1. RPC style
2. Document style



Difference between RPC and Document web services

There are many differences between RPC and Document web services. The important differences between RPC and Document are given below:

RPC Style

1) RPC style web services use method name and parameters to generate XML structure.
2) The generated WSDL is **difficult to be validated** against schema.
3) In RPC style, SOAP **message is sent as many elements**.
4) RPC style message is **tightly coupled**.
5) In RPC style, SOAP message **keeps the operation name**.
6) In RPC style, parameters are sent as **discrete values**.

Document Style

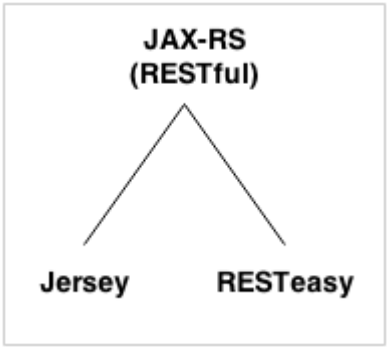
1) Document style web services **can be validated against predefined schema**.
2) In document style, SOAP message is **sent as a single document**.
3) Document style message is **loosely coupled**.
4) In Document style, SOAP message **loses the operation name**.
5) In Document style, parameters are sent in **XML format**.

JAX-RS Tutorial

JAX-RS provides concepts and examples of JAX-RS API. This is designed for beginners and professionals.

There are two main implementation of JAX-RS API.

1. Jersey
2. RESTEasy



JAX-RS Annotations

The **javax.ws.rs** package contains JAX-RS annotations.

Annotation	Description
Path	It identifies the URI path. It can be specified on class or method.
PathParam	represents the parameter of the URI path.
GET	specifies method responds to GET request.
POST	specifies method responds to POST request.
PUT	specifies method responds to PUT request.
HEAD	specifies method responds to HEAD request.
DELETE	specifies method responds to DELETE request.
OPTIONS	specifies method responds to OPTIONS request.
FormParam	represents the parameter of the form.
QueryParam	represents the parameter of the query string of an URL.
HeaderParam	represents the parameter of the header.
CookieParam	represents the parameter of the cookie.
Produces	defines media type for the response such as XML, PLAIN, JSON etc. It defines the media type that the methods of a resource class or MessageBodyWriter can produce.
Consumes	It defines the media type that the methods of a resource class or MessageBodyReader can produce.

6. Creating, Publishing, Testing and Describing a Web services (WSDL)

WSDL publish and test a HugeInteger web service that performs calculations with positive integers up to 100 digits long (maintained as arrays of digits). Such integers are much larger than Java‘s integral primitive types can represent. The HugeInteger web service provides methods that take two ―huge integers‖ (represented as Strings) and determine their sum, their difference, which is larger, which is smaller or whether the two numbers are equal.

1. Creating a Web Application Project and Adding a Web Service Class in Netbeans

When you create a web service in Netbeans, you focus on the web service‘s logic and let the IDE handle the web service‘s infrastructure. To create a web service in Netbeans, you first create a **Web Application** project. Netbeans uses this project type for web services that are invoked by other applications.

➢ ***Creating a*** *Web Application* ***Project in Netbeans*** To create a web application, perform the following steps:

1. Select File > New Project to open the New Project dialog.
2. Select Web from the dialog‘s Categories list, then select Web Application from the Projectslist. Click Next >.
3. Specify the name of your project (HugeInteger) in the Project Name field and specify where you‘d like to store the project in the Project Location field. You can click theBrowse button to select the location.

1 Select Sun Java System Application Server 9 from the Server drop-down list.

2 Select Java EE 5 from the J2EE Version drop-down list.

6. Click Finish to dismiss the New Project dialog.

This creates a web application that will run in a web browser.

➢ ***Adding a Web Service Class to a*** *Web Application* ***Project*** Perform the following steps to add a web service class to the project:

1.In the Projects tab in Netbeans, right click the HugeInteger project‘s node and se-lectNew > Web Service… to open the New Web Service dialog.

2. Specify HugeInteger in the Web Service Name field.
3. Specify com.deitel.iw3htp4.ch28.hugeinteger in the Package field.
4. Click Finish to dismiss the New Web Service dialog.

The IDE generates a sample web service class with the name you specified in *Step 2*

2. Defining the HugeInteger Web Service in Netbeans

The following example shows the HugeInteger web service‘s code. You can implement this code yourself in the HugeInteger.java file created in Section 28.3.1, or you can simply replace the code inHugeInteger.java with a copy of our code from this example‘s folder. You can find this file in the project‘s src\java\com\deitel\iw3htp4\ch28\hugeinteger folder. // HugeInteger.java

```
// HugeInteger web service that performs operations on large integers.

package com.deitel.iw3htp4.ch28.hugeinteger; import
javax.jws.WebService; import javax.jws.WebMethod;
import javax.jws.WebParam @WebService( // annotates the
class as a web service name = "HugeInteger", // sets
class name
        serviceName = "HugeIntegerService" ) // sets the service name
public class HugeInteger
  {
      private final static int MAXIMUM = 100;
public int[] number = new int[ MAXIMUM ];
public String toString()
   {
         String value = "";
         // convert HugeInteger to a String
for ( int digit : number )
```

```
                value = digit + value; // places next digit at beginning of value
                  // locate position of first non-zero digit
int length = value.length();
               int position = -1;
                 for ( int i = 0; i < length; i++ )
                 {
                      if ( value.charAt( i ) != '0' )
                  {
                 position = i; // first non-zero digit
                          break;
                     }
               } // end for

                 return ( position != -1 ? value.substring( position ) : "0" );
           } // end method toString
          // creates a HugeInteger from a String
           public static HugeInteger parseHugeInteger( String s )
         {
                  HugeInteger temp = new HugeInteger();
int size = s.length();                for ( int i = 0; i < size;
i++ )
                   temp.number[ i ] = s.charAt( size - i - 1 ) - '0';

             return temp;
         } // end method parseHugeInteger
              // WbMethod that adds huge integers represented by String arguments
         @WebMethod( operationName = "add" )
         public String add( @WebParam( name = "first" ) String first,
            @WebParam( name = "second" ) String second )
     {
               int carry = 0; // the value to be carried
               HugeInteger operand1 = HugeInteger.parseHugeInteger( first );
               HugeInteger operand2 = HugeInteger.parseHugeInteger( second );
               HugeInteger result = new HugeInteger(); // stores addition result
                        // perform addition on each digit
                for ( int i = 0; i < MAXIMUM; i++ )
               {

                   // add corresponding digits in each number and the carried value;
// store result in the corresponding column of HugeInteger result  result.number[ i ]
=  ( operand1.number[ i ] + operand2.number[ i ] + carry ) % 10;

                   // set carry for next column
```

```
                carry =
                    ( operand1.number[ i ] + operand2.number[ i ] + carry ) / 10;
            }       //      end      for
return result.toString();
        } // end WebMethod add
       // WebMethod that subtracts integers represented by String arguments
        @WebMethod( operationName = "subtract" )
        public String subtract( @WebParam( name = "first" ) String first,
            @WebParam( name = "second" ) String second )
      {
            HugeInteger operand1 = HugeInteger.parseHugeInteger( first );
          HugeInteger operand2 = HugeInteger.parseHugeInteger( second );
            HugeInteger result = new HugeInteger(); // stores difference
// subtract bottom digit from top digit

          for ( int i = 0; i < MAXIMUM; i++ )
        {
              // if the digit in operand1 is smaller than the corresponding

            // digit in operand2, borrow from the next digit  if
( operand1.number[ i ] < operand2.number[ i ] )
      operand1.borrow( i );

    // subtract digits
       result.number[ i ] = operand1.number[ i ] - operand2.number[ i ];
} // end for
    return result.toString();

    }  //  end  WebMethod  subtract
// borrow 1 from next digit
      private void borrow( int place )
      {
           if ( place >= MAXIMUM )                  throw new
IndexOutOfBoundsException();             else if ( number[ place
+ 1 ] == 0 ) // if next digit is zero                borrow( place + 1
); // borrow from next digit             number[ place ] += 10; //
add 10 to the borrowing digit

           --number[ place + 1 ]; // subtract one from the digit to the left
       } // end method borrow
      // WebMethod that returns true if first integer is greater than second
      @WebMethod( operationName = "bigger" )
       public boolean bigger( @WebParam( name = "first" ) String first,
            @WebParam( name = "second" ) String second )
```

```
        {
            try // try subtracting first from second
            {
                String difference = subtract( first, second );
return !difference.matches( "^[0]+$" );
            } // end try

            catch ( IndexOutOfBoundsException e ) // first is less than second
            {
return false;
            } // end catch
        } // end WebMethod bigger
        // WebMethod that returns true if the first integer is less than second
        @WebMethod( operationName = "smaller" )
        public boolean smaller( @WebParam( name = "first" ) String first,
            @WebParam( name = "second" ) String second )
        {
            return bigger( second, first );
        } // end WebMethod smaller
        // WebMethod that returns true if the first integer equals the second

        @WebMethod( operationName = "equals" )
        public boolean equals( @WebParam( name = "first" ) String first,
            @WebParam( name = "second" ) String second )
        {
            return !( bigger( first, second ) || smaller( first, second ) );
        } // end WebMethod equals
    } // end class HugeInteger
```

3. **Publishing the HugeInteger Web Service from Netbeans** we‘ve created the HugeInteger web service class, we‘ll use Netbeans to build and publish (i.e., deploy) the web service so that clients can consume its services. Netbeans handles all the details of building and deploying a web service for you. This includes cre-ating the framework required to support the web service. Right click the project name (HugeInteger) in the NetbeansProjects tab to display the pop-up menu shown in following fig. To determine if there are any compilation errors in your project, select the Build Project option. When the project compiles successfully, you can select Deploy Project to deploy the project to the server If the code in the project has changed since the last build, selecting Deploy Project also builds the project.

Selecting Run Project executes the web application. If the web application was not previously built or deployed, this option performs these tasks first. Note that both the Deploy Project and Run Project options also start the application server (in our case Sun Java System Application Server) if it is not already running.

4. **Testing the HugeInteger Web Service with Sun Java System Application Server‟s Tester Web page**

This server can dynamically

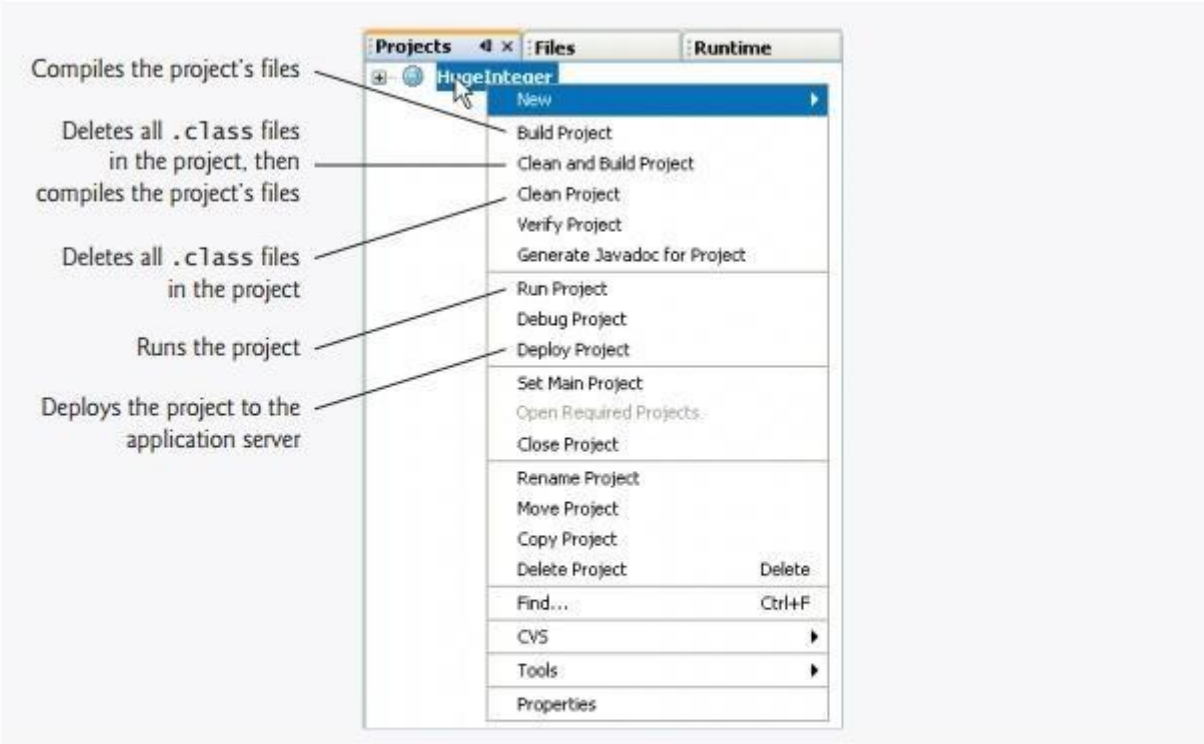


Fig. 28.3 | Pop-up menu that appears when you right click a project name in the Netbeans **Projects** tab.

create a web page for testing a web service‘s methods from a web browser. To enable this capability:

1. Right click the project name (HugeInteger) in the Netbeans Projects tab and select Propertiesfrom the pop-up menu to display the Project Properties dialog.
2. Click Run under Categories to display the options for running the project.
3. In the Relative URL field, type /HugeIntegerService?Tester.
4. Click OK to dismiss the Project Properties dialog.

To test HugeInteger‘s web methods, type two positive integers into the text fields to the right of a particular method‘s button, then click the button to invoke the web method and see the result.

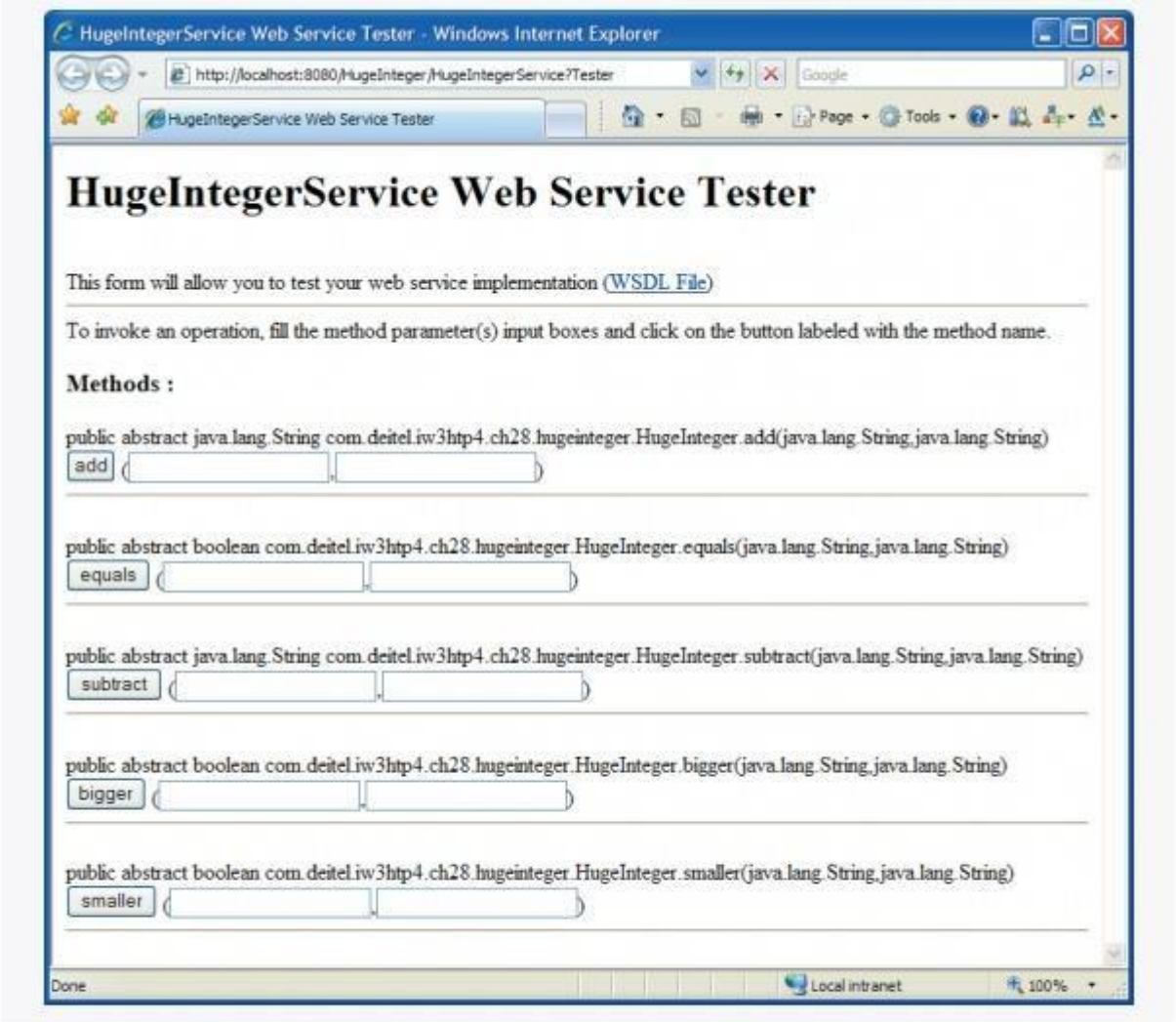


Fig. 28.4 | `Tester` web page created by Sun Java System Application Server for the `HugeInteger` web service.

a) Invoking the `HugeInteger` web service's `add` method.

Fig. 28.5 | Testing `HugeInteger`'s `add` method. (Part 1 of 2.)

7. Consuming a Web Service

A web service client can be any type of application or even another web service. You enable a client application to consume a web service by **adding a web service reference** to

theapplication. This process defines the proxy class that allows the client to access the web service.

➢ **Creating a Client in Netbeans to Consume the** HugeInteger **Web Service**

When you add the web service reference, the IDE creates and compiles the **client-side ar-tifacts**—the framework ofJava code that supports the client-side proxy class. The client then calls methods on an object of the proxy class, which uses the rest of the artifacts to interact with the web service.

Creating a Desktop Application Project in Netbeans

Before performing the steps in this section, ensure that the HugeInteger web service has been deployed and that the Sun Java System Application Server is running (see Section 28.3.3).

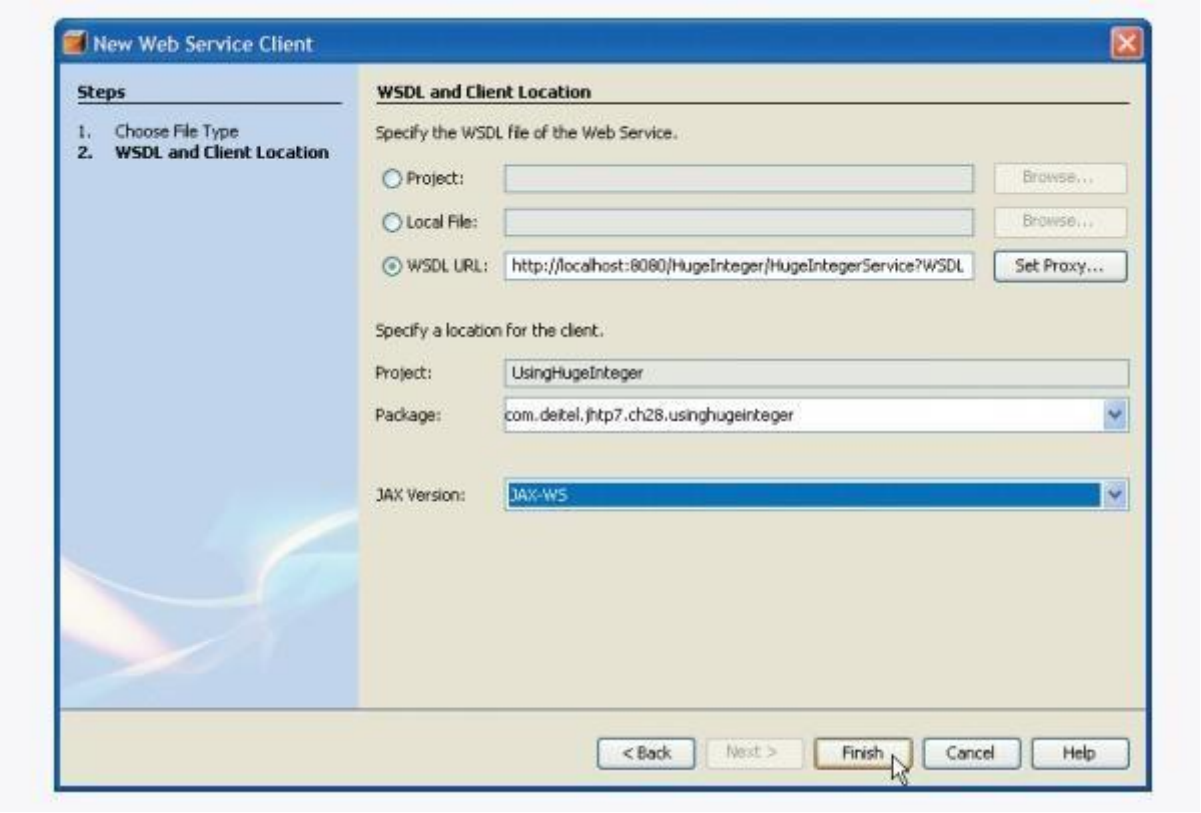
Perform the following steps to create a client Java desktop application in Netbeans:

1. Select File > New Project… to open the New Project dialog.
2. Select General from the Categories list and Java Application from the Projects list, then click Next >.
3. Specify the name *UsingHugeInteger* in the Project Name field and uncheck the Create Main Class checkbox. In a moment, you'll add a subclass of JFrame that contains a mainmethod.
4. Click Finish to create the project.

Adding a Web Service Reference to an Application

Next, you'll add a web service reference to your application so that it can interact with theHugeInteger web service. To add a web service reference, perform the following steps.

1. Right click the project name (UsingHugeInteger) in the Netbeans Projects tab.
2. Select New > Web Service Client… from the pop-up menu to display the New Web Service Client dialog (Fig. 28.7).



3. In the field, specify the URL WSDL URL http://localhost:8080/HugeInteger/

HugeIntegerService?WSDL (Fig. 28.7). This URL tells the IDE where to find the web service's WSDL description. [Note: If the Sun Java System Application Serv-er is located on a different computer, replace localhost with the hostname or IP address of that computer.] The IDE uses this WSDL description to generate the client-side artifacts that compose and support the proxy. Note that the New Web Service Client dialog enables you to search for web services in several locations. Many companies simply distribute the exact WSDL URLs for their web services, which you can place in the field.

6 In the Package field, specify com.deitel.iw3htp4.ch28.usinghugeinteger as the package name.

7 Click Finish to dismiss the New Web Service Client dialog.

In the Netbeans Projects tab, the UsingHugeInteger project now contains a Web Service Referencesfolder with the HugeInteger web service's proxy (Fig. 28.8). Note that the proxy's name is listed asHugeIntegerService, as we specified in line 11 of Fig. 28.2.

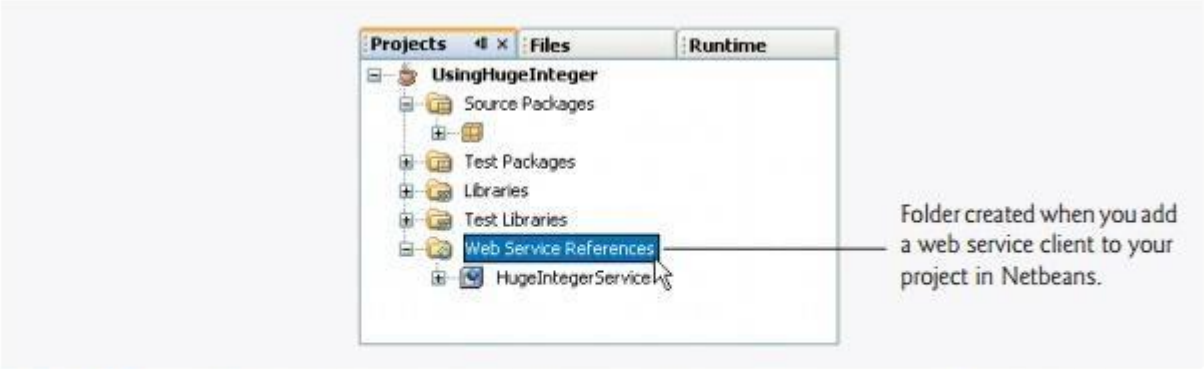


Fig. 28.8 | Netbeans **Project** tab after adding a web service reference to the project.

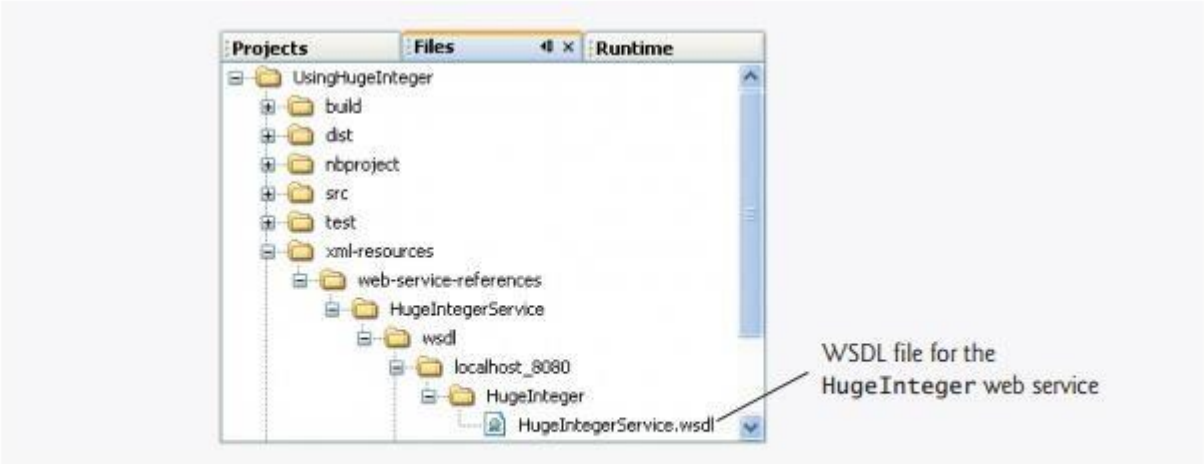


Fig. 28.9 | Locating the `HugeIntegerService.wsdl` file in the Netbeans **Files** tab.

When you specify the web service you want to consume, Netbeans accesses the web service's WSDL information and copies it into a file in your project (named HugeInte-gerService.wsdl in thisexample). You can view this file from the Netbeans Files tab by expanding the nodes in theUsingHugeInteger project's xml-resources folder as shown in Fig. 28.9. If the web service changes, the client-side artifacts and the client's copy of the WSDL file can be regenerated by right clicking the HugeIntegerService node shown in Fig. 28.8 and selecting Refresh Client.

You can view the IDE-generated client-side artifacts by selecting the Netbeans Files tab and expanding the UsingHugeInteger project‘s build folder as shown in Fig. 28.10.

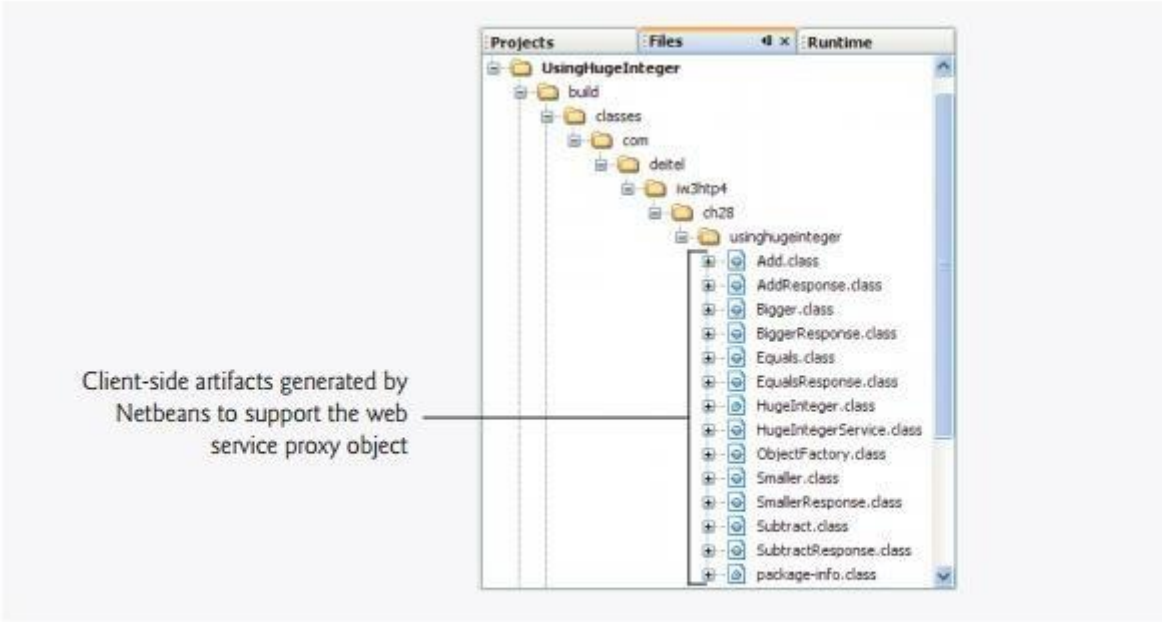


Fig. 28.10 | Viewing the `HugeInteger` web service's client-side artifacts generated by Netbeans.

8. **<u>Database Driven web service from an application</u>** web-based businesses are becoming increasingly prevalent, it is common for web applications to consume web services.

1. Configuring Java DB in Netbeans and Creating the Reservation **Database** web service uses a Reservation database containing a single table namedSeats to locate a seat matching a client‘s request. To build the Reservation data-base, review the steps presented in Section 27.2.1 for building the AddressBook database. This chapters examples directory contains a SQL script to build the Seats table and pop-ulate it with sample data. The sample data is shown in Fig. 28.15.

Number	Location	Class	Taken
1	Aisle	Economy	0
2	Aisle	Economy	0
3	Aisle	First	0
4	Middle	Economy	0
5	Middle	Economy	0
6	Middle	First	0
7	Window	Economy	0
8	Window	Economy	0
9	Window	First	0
10	Window	First	0

Fig. 28.15 | Seats table's data.

Creating the Reservation Web Service

You can now create a web service that uses the Reservation database (Fig. 28.16). The airline reservation web service has a single web method—reserve (lines 26–78)—which searches theSeats table to locate a seat matching a user‘s request. The method takes two arguments— a Stringrepresenting the desired seat type (i.e., "Window", "Middle" or "Aisle") and a String representingthe desired class type (i.e., "Economy" or "First"). If it finds an appropriate seat, method reserveupdates the database to make the reservation and returns true; otherwise, no reservation is made, and the method returns false.

Our database contains four columns—the seat number (i.e., 1–10), the seat type (i.e., Window,Middle or Aisle), the class type (i.e., Economy or First) and a column containing either 1 (true) or0 (false) to indicate whether the seat is taken. Lines 34–39 retrieve the seat numbers of any available seats matching the requested seat and class type. This state-ment fills the resultSet with the results of the query

```
SELECT "NUMBER" FROM "SEATS"
WHERE ("TAKEN" = 0) AND ("TYPE" = type) AND ("CLASS" = class)
```

The parameters *type* and *class* in the query are replaced with values of method reserve‘s seatType andclassType parameters. When you use the Netbeans tools to create a data-base table and its columns, the Netbeans tools automatically place the table and column names in double quotes.

If resultSet is not empty (i.e., at least one seat is available that matches the selected criteria), the condition in line 42 is true and the web service reserves the first matching seat number. Recall that ResultSet method next returns true if a nonempty row exists, and positions the cursor on that row

```
 UPDATE "SEATS"
 SET "TAKEN" = 1
WHERE ("NUMBER" = number)
```

➢ **Creating a Web Application to Interact with the** Reservation**Web Service**

```
<?xml version="1.0" encoding="UTF-8"?>
    <!-- Fig. 28.17 Reserve.jsp -->
   <!-- JSP that allows a user to select a seat -->
  <jsp:root version="1.2"
       xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:jsp="http://java.sun.com/JSP/Page"
       xmlns:webuijsf="http://www.sun.com/webui/webuijsf">
<jsp:directive.page contentType="text/html;charset=UTF-8"
pageEncoding="UTF-8"/>

<f:view>
      <webuijsf:page binding="#{Reserve.page1}" id="page1">
       <webuijsf:html binding="#{Reserve.html1}" id="html1">
      <webuijsf:head          binding="#{Reserve.head1}"          id="head1">
<webuijsf:link binding="#{Reserve.link1}" id="link1"
     url="/resources/stylesheet.css"/>
     </webuijsf:head>
      <webuijsf:body binding="#{Reserve.body1}" id="body1"

       style="-rave-layout: grid">
      <webuijsf:form binding="#{Reserve.form1}" id="form1">
<webuijsf:label binding="#{Reserve.instructionLabel}"
id="instructionLabel" style="left: 24px; top: 24px;
position: absolute" text="Please select the seat type       and
class to reserve:"/>
     <webuijsf:dropDown                              binding="#{Reserve.seatTypeDropDown}"
id="seatTypeDropDown" items=
      "#{Reserve.seatTypeDropDownDefaultOptions.options}"
style="left: 310px; top: 21px; position: absolute"
      valueChangeListenerExpression=
      "#{Reserve.seatTypeDropDown_processValueChange}"/>
<webuijsf:dropDown       binding="#{Reserve.classTypeDropDown}"
id="classTypeDropDown" items=
      "#{Reserve.classTypeDropDownDefaultOptions.options}"
style="left: 385px; top: 21px; position: absolute"
     valueChangeListenerExpression=
      "#{Reserve.classTypeDropDown_processValueChange}"/>
     <webuijsf:button actionExpression=
      "#{Reserve.reserveButton_action}" binding=
     "#{Reserve.reserveButton}" id="reserveButton" style=
"height: 20px; left: 460px; top: 21px; position:
absolute; width: 100px" text="Reserve"/>
```

```
<webuijsf:label binding="#{Reserve.errorLabel}"
id="errorLabel" rendered="false" style="color: red;       left:
24px; top: 48px; position: absolute" text="This       type of
seat is not available. Please modify your       request and try
again."/>
      <webuijsf:label  binding="#{Reserve.successLabel}"
id="successLabel"   rendered="false"   style="left:   24px;
top: 24px; position: absolute"
      text="Your reservation has been made. Thank you!"/>
     </webuijsf:form>
      </webuijsf:body>
       </webuijsf:html>
   </webuijsf:page>
     </f:view>
    </jsp:root>
```

Fig. 28.17 | JSP that allows a user to select a seat.

9. SOAP

SOAP (Simple Object Access Protocol) is a platform-independent protocol that uses XML to facilitate remote procedure calls, typically over HTTP. SOAP is one common protocol for passing information between web service clients and web services. The protocol that transmits request-andresponse messages is also known as the web service‘s **wire format** or **wire protocol**, because it defines how information is sent ―along the wire.‖

Each request and response is packaged in a **SOAP message** (also known as a **SOAPenvelope**)— an XML ―wrapper‖ containing the information that a web service requires to process the message. SOAP messages are written in XML so that they are platform inde-pendent. Many**firewalls**—security barriers that restrict communication among net-works—are configured to allow HTTP traffic to pass through so that clients can browse websites on web servers behind firewalls. Thus, XML and HTTP enable computers on dif-ferent platforms to send and receive SOAP messages with few limitations.

The wire format used to transmit requests and responses must support all data types passed between the applications. Web services also use SOAP for the many data types it supports. SOAP supports primitive types (e.g., int) and their wrapper types (e.g., Integer), as well asDate, Time and others. SOAP can also transmit arrays and objects of user-defined types

When a program invokes a web method, the request and all relevant information are packaged in a SOAP message and sent to the server on which the web service resides. The web service processes the SOAP message‘s contents (contained in a SOAP envelope), which specify the method that the client wishes to invoke and the method‘s arguments. This process of interpreting a SOAP message‘s contents is known as **parsing a SOAP mes-sage**. After the webservice receives and parses a request, the proper method is called with any specified arguments, and the response is sent back to the client in another SOAP mes-sage. The client-side proxy parses the response, which contains the result of the method call, and returns the result to the client application.

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:
    <soapenv:Body>
        <ns1:add>
            <first>99999999999999999</first>
            <second>1</second>
        </ns1:add>
    </soapenv:Body>
</soapenv:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:
    <soapenv:Body>
        <ns1:addResponse>
            <return>100000000000000000</return>
        </ns1:addResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Fig:SOAP messages for the HugeInteger web service‘s

R15

Code: 15A05605

B.Tech III Year II Semester (R15) Regular Examinations May/June 2018

WEB & INTERNET TECHNOLOGIES

(Computer Science & Engineering)

Time: 3 hours Max. Marks: 70

PART – A

(Compulsory Question)

*****

1 Answer the following: (10 X 02 = 20 Marks)

(a) What are the different types of lists in HTML?
(b) What is the purpose of meta tags in HTML?
(c) Define application wide error page in JSP.
(d) List the different types of statements in JDBC.
(e) What do you mean by casting of data types in PHP?
(f) How to pass a variable by reference in PHP?
(g) List the applications of XML.
(h) Why XML uses tree based parsing and streaming?
(i) What is synchronous request in AJAX?
(j) Are AJAX applications easier to develop than traditional web applications?

PART – B

(Answer all five units, 5 X 10 = 50 Marks)

UNIT – I

2 What is HTML table? Explain table element with necessary attributes. Write the HTML code to print following table.

<table>
<tr><td colspan="2">ONE</td><td>TWO</td></tr>
<tr><td rowspan="2">THREE</td><td>FOUR</td><td rowspan="2">SIX</td></tr>
<tr><td>FIVE</td></tr>
<tr><td>SEVEN</td><td colspan="2">EIGHT</td></tr>
</table>

OR

3 What do you mean by “class” and “id” in CSS? Show the use of external CSS with example.

UNIT – II

4 Write a java script program which reads number and check whether that number is strong number or not.

OR

5 Write a script that asks the user to enter two numbers and outputs text that displays the sum, product, difference and quotient of the two numbers.

UNIT – III

6 Write a PHP program to store user registration information (like user name, address, DOB, age, Aadhar card number, gender) with data base and display that details on view screen in tabular manner.

OR

7 Write a PHP script to sort the elements in descending order of an array.

UNIT – IV

8 Design an XML schema for university information management.

OR

9 Explain the working of XML processor in detail. Also mention the purpose of XML program.

UNIT – V

10 Describe the integration of PHP and AJAX with an example.

OR

11 Discuss the role of WSDL web services in AJAX.

*****

R15

Code: 15A05605

B.Tech III Year II Semester (R15) Supplementary Examinations December/January 2018/19

WEB & INTERNET TECHNOLOGIES

(Computer Science & Engineering)

Time: 3 hours Max. Marks: 70

PART – A

(Compulsory Question)

*****

1 Answer the following: (10 X 02 = 20 Marks)
(a) Write the structure of HTML program.
(b) Write the characteristics of DHTML.
(c) What are the differences between custom JSP and Servlets?
(d) What are the scoping rules for the java script?
(e) List the statements that are used to connect PHP with MySQL.
(f) How to define a variable accessible in function of PHP script?
(g) What is PCDATA in XML?
(h) What is document type definition (DTD)?
(i) Differentiate between proxied and proxy less calls in AJAX.
(j) What is script manager in AJAX?

PART – B

(Answer all five units, 5 X 10 = 50 Marks)

UNIT – I

2 Explain following HTML tags with their important attributes.
(i) <form>. (ii) <frame net> and <frame>. (iii) <img>.

OR

3 Draw the neat block diagram and explain the CSS box model.

UNIT – II

4 Design a form using HTML and java script to collect the details from the user such as name, gender (radio button), address, state & city. Create a dynamic drop down list for state and based on state, create another drop down list containing list of cities for that state.

OR

5 Write a script that reads an integer and determines whether it is arm strong number or not.

UNIT – III

6 Write a PHP program that receives the value of N using HTML form and displays the first N Fibonacci numbers as HTML list.

OR

7 Write PHP code to upload the image file.

UNIT – IV

8 Explain how XML is useful in defining data for web applications.

OR

9 Explain various types of XML schema data types used.

UNIT – V

10 Is AJAX code cross browser compatible? Justify with an example.

OR

11 Explain the SOAP web services in AJAX.

*****