WEB TECHNOLOGIES

UNIT-1

What is the Internet?

The Internet is a worldwide network of computer networks that connects university, government, commercial, and other computers in over 150 countries. There are thousands of networks, tens of thousands of computers, and millions of users on the Internet, with the numbers expanding daily. Using the Internet, you can send electronic mail, chat with colleagues around the world, and obtain information on a wide variety of subjects.

Three principal uses of the Internet are:

- **Electronic mail**. Electronic mail, or e-mail, lets you electronically "mail" messages to users who have Internet E-mail addresses. Delivery time varies, but it's possible to send mail across the globe and get a response in minutes. LISTSERVs are special interest mailing lists which allow for the exchange of information between large numbers of people.
- **USENET newsgroups**. USENET is a system of special interest discussion groups, called newsgroups, to which readers can send, or "post" messages which are then distributed to other computers in the network. (Think of it as a giant set of electronic bulletin boards.) Newsgroups are organized around specific topics, for example, **alt.education.research, alt.education.distance, and misc.education.science**.
- **Information files**. Government agencies, schools, and universities, commercial firms, interest groups, and private individuals place a variety of information on-line. The files were originally text only, but increasingly contain pictures and sound.

How Do I Explore the Internet?

To access the Internet, you'll need a personal computer, a modem (or direct link to a network), telecommunications software, a telephone line, and an Internet account. Don't worry--this is easier than it sounds, but it still helps when you're getting started to have a few good books on the subject, or better yet, a friend who's an experienced "cyber surfer." Many universities provide Internet accounts to their faculty and students at little or no cost. Commercial vendors will provide Internet service for a fee. Make sure that you access your Internet provider with a local telephone call--otherwise, long distance charges will apply.

The easiest way for new users to navigate the Internet may be through the "gopher," a navigational system that uses a series of menus to organize and provide access to information. Unfortunately, "gopher," while easy to use, provides text-only information. It is much more

rewarding to take full advantage of the multimedia opportunities available on the World-Wide Web (WWW). This system organizes information to provide for linkages to related documents (hypertext links), which allow users to move quickly and easily to related documents.

Software such as Mosaic and Netscape give users a graphical interface and (theoretically) allow for effortless "point and click" travel through cyberspace. If you want to use programs such as Mosaic and Netscape, you will need an up-to-date personal computer and a fast modem--the faster the better--but most users find the rewards worth the extra investment.

What is Hypertext?

Hypertext is a powerful cross-referencing tool meant for user-driven access to an ocean wealth of interconnected information either static or dynamic in an electronic format. Simply put, hypertext may refer to plain simple text that contains links to access other chunks of text within the same or different document. It provides a means to organize and present information in a way that is easily accessible to the end users. It's more like a user-driven tool to represent textual information which are linked together to provide more flexibility and a greater degree of control. This allows users or readers to move from one location to another via hyperlinks or "go to" links. Links connect nodes to other documents and are usually activated when clicked upon by a mouse or other pointing device.

What is Hypermedia?

Hypermedia is an extension of hypertext that employs multiple forms of media such as text, graphics, audio or video sequences, still or moving graphics, etc. The structure of hypermedia is quite similar to that of a hypertext, except it's not constrained to be just text-based. It extends the capabilities of hypertext systems by creating clickable links within web pages to create a network of interconnected non-linear information which user can both access and interact with for a better multimedia experience. The most common hypermedia type is image links which are often linked to other web pages. It is used in a variety of applications from problem solving and qualitative research to electronic studying and sophisticated learning.

Difference between Hypertext and Hypermedia

Definition

Hypertext simply refers to text that contains links to other chunks of text to which the user is transferred to usually by a mouse click or keypress. The documents are linked together via hyperlinks which allow users to jump from one document to another within the same or different web pages. Hypermedia, on the other hand, is an extension of the term hypertext used in a similar way except it's not constrained to text elements. In fact, hypermedia contains different media elements or morphologies such as audio, images, video, and still or moving graphics.

Representation

Hypertext is an interconnected network of documents and other media referenced through links between them. It can contain either static or dynamic content in an electronic format. The static content is the content that can be delivered directly to the end users without any modification whereas dynamic content may subject to change based on user inputs. Hypermedia is the next level of multimedia experience which extends the notion of hypertext links to include not only text but a wide range of other multimedia elements such as audio, video, and graphics.

Technology

Although the term hypertext is widely used in association with the World Wide Web, the technology has been around since ages. The hypertext technology is solely based on human-computer interaction by strong cross referencing tools called hyperlinks. It facilitates effective use of text and links and how to implement it on the World Wide Web. Hypermedia technology is based on non-linear forms of media which include not only plain text but also other multimedia elements to enhance your overall multimedia experience. Hypermedia technology is a major breakthrough in the field of education.

HTTP - HyperText Transfer Protocol

HTTP means **H**yper**T**ext **T**ransfer **P**rotocol. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page. The other main standard that controls how the World Wide Web works is HTML, which covers how Web pages are formatted and displayed.

HTTP is a Stateless Protocol

HTTP is called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it. This is the main reason that it is difficult to implement Web sites that react intelligently to user input. This shortcoming of HTTP is being addressed in a number of new technologies, including ActiveX, Java, JavaScript and cookies.

HTTP Status Codes are Error Messages

Errors on the Internet can be quite frustrating — especially if you do not know the difference between a 404 error and a 502 error. These error messages, also called HTTP status codes are response codes given by Web servers and help identify the cause of the problem.
For example, "404 File Not Found" is a common HTTP status code. It means the Web server cannot find the file you requested. This means the webpage or other document you tried to load in your Web browser has either been moved or deleted, or you entered the wrong URL or document name.

Knowing the meaning of the HTTP status code can help you figure out what went wrong. On a 404 error, for example, you could look at the URL to see if a word looks misspelled, then correct it and try it again. If that doesn't work, backtrack by deleting information between each backslash, until you come to a page on that site that isn't a 404. From there you may be able to find the page you're looking for.
Additional information on HTTP error codes can be found in Webopedia's common HTTP status codes article.

Custom 404 Error Pages
Many websites create custom 404 error pages that will help users locate a valid page or document within the website. For example, if you land on a 404 File Not Found page via Webopedia.com, a custom error page will load providing quick links to on-site navigation and site search features to help you find what you were looking for.

What about HTTPS?
A similar abbreviation, **HTTPS** means **H**yper **T**ext **T**ransfer **P**rotocol **S**ecure. Basically, it is the secure version of HTTP. Communications between the browser and website are encrypted by Transport Layer Security (TLS), or its predecessor, Secure Sockets Layer (SSL).

HTML

HTML (HyperText Markup Language) is the most basic building block of the Web. It defines the meaning and structure of web content. Other technologies besides HTML are generally used to describe a web page's appearance/presentation (CSS) or functionality/behavior (JavaScript). "Hypertext" refers to links that connect web pages to one another, either within a single website or between websites. Links are a fundamental aspect of the Web. By uploading content to the Internet and linking it to pages created by other people, you become an active participant in the World Wide Web.

HTML uses "markup" to annotate text, images, and other content for display in a Web browser.
HTML markup includes special "elements" such
as <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img >, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <v ideo>, <ul>, <ol>, <li> and many others.
An HTML element is set off from other text in a document by "tags", which consist of the element name surrounded by "<" and ">". The name of an element inside a tag is case insensitive. That is, it can be written in uppercase, lowercase, or a mixture. For example, the <title> tag can be written as <Title>, <TITLE>, or in any other way.

URLs

What Is a URL?

If you've been surfing the Web, you have undoubtedly heard the term URL and have used URLs to access HTML pages from the Web.

It's often easiest, although not entirely accurate, to think of a URL as the name of a file on the World Wide Web because most URLs refer to a file on some machine on the network. However, remember that URLs also can point to other resources on the network, such as database queries and command output.

Definition:

URL is an acronym for *Uniform Resource Locator* and is a reference (an address) to a resource on the Internet.

A URL has two main components:

- Protocol identifier: For the URL http://example.com, the protocol identifier is http.
- Resource name: For the URL http://example.com, the resource name is example.com.

Note that the protocol identifier and the resource name are separated by a colon and two forward slashes. The protocol identifier indicates the name of the protocol to be used to fetch the resource. The example uses the Hypertext Transfer Protocol (HTTP), which is typically used to serve up hypertext documents. HTTP is just one of many different protocols used to access different types of resources on the net. Other protocols include File Transfer Protocol (FTP), Gopher, File, and News.

The resource name is the complete address to the resource. The format of the resource name depends entirely on the protocol used, but for many protocols, including HTTP, the resource name contains one or more of the following components:

Host Name

The name of the machine on which the resource lives.

Filename

The pathname to the file on the machine.

Port Number

The port number to which to connect (typically optional).

Reference

> A reference to a named anchor within a resource that usually identifies a specific location within a file (typically optional).

For many protocols, the host name and the filename are required, while the port number and reference are optional. For example, the resource name for an HTTP URL must specify a server on the network (Host Name) and the path to the document on that machine (Filename); it also can specify a port number and a reference.

Creating a URL

The easiest way to create a URL object is from a String that represents the human-readable form of the URL address. This is typically the form that another person will use for a URL. In your Java program, you can use a String containing this text to create a URL object:

```
URL myURL = new URL("http://example.com/");
```

The URL object created above represents an *absolute URL*. An absolute URL contains all of the information necessary to reach the resource in question. You can also create URL objects from a *relative URL* address.

Creating a URL Relative to Another

A relative URL contains only enough information to reach the resource relative to (or in the context of) another URL.

Relative URL specifications are often used within HTML files. For example, suppose you write an HTML file called JoesHomePage.html. Within this page, are links to other pages, PicturesOfMe.html and MyKids.html, that are on the same machine and in the same directory as JoesHomePage.html. The links to PicturesOfMe.html and MyKids.html from JoesHomePage.html could be specified just as filenames, like this:

```
<a href="PicturesOfMe.html">Pictures of Me</a>
<a href="MyKids.html">Pictures of My Kids</a>
```

These URL addresses are *relative URLs*. That is, the URLs are specified relative to the file in which they are contained — JoesHomePage.html.

In your Java programs, you can create a URL object from a relative URL specification. For example, suppose you know two URLs at the site example.com:

```
http://example.com/pages/page1.html
http://example.com/pages/page2.html
```

You can create URL objects for these pages relative to their common base URL: http://example.com/pages/ like this:

```
URL myURL = new URL("http://example.com/pages/");
URL page1URL = new URL(myURL, "page1.html");
URL page2URL = new URL(myURL, "page2.html");
```

This code snippet uses the URL constructor that lets you create a URL object from another URL object (the base) and a relative URL specification. The general form of this constructor is:

URL(URL *baseURL*, String *relativeURL*)

The first argument is a URL object that specifies the base of the new URL. The second argument is a String that specifies the rest of the resource name relative to the base. If baseURL is null, then this constructor treats relativeURL like an absolute URL specification. Conversely, if relativeURL is an absolute URL specification, then the constructor ignores baseURL.

This constructor is also useful for creating URL objects for named anchors (also called references) within a file. For example, suppose the page1.html file has a named anchor called BOTTOM at the bottom of the file. You can use the relative URL constructor to create a URL object for it like this:

```
URL page1BottomURL = new URL(page1URL,"#BOTTOM");
```

A **media type** (also known as a **Multipurpose Internet Mail Extensions or MIME type**) is a standard that indicates the nature and format of a document, file, or assortment of bytes. It is defined and standardized in IETF's RFC 6838.
The Internet Assigned Numbers Authority (IANA) is responsible for all official MIME types, and you can find the most up-to-date and complete list at their Media Types page.

Structure of a MIME type

The simplest MIME type consists of a *type* and a *subtype*; these are each strings which, when concatenated with a slash (/) between them, comprise a MIME type. No whitespace is allowed in a MIME type:

```
type/subtype
```

The ***type*** represents the general category into which the data type falls, such as video or text. The ***subtype*** identifies the exact kind of data of the specified type the MIME type represents. For example, for the MIME type text, the subtype might be plain (plain text), html (HTML source code), or calendar (for iCalendar/.ics) files.

Each type has its own set of possible subtypes, and a MIME type always has both a type and a subtype, never just one or the other.

An optional **parameter** can be added to provide additional details:

> *type*/*subtype*;*parameter=value*

For example, for any MIME type whose main type is text, the optional charset parameter can be used to specify the character set used for the characters in the data. If no charset is specified, the default is ASCII (US-ASCII) unless overridden by the user agent's settings. To specify a UTF-8 text file, the MIME type text/plain;charset=UTF-8 is used.
MIME types are case-insensitive but are traditionally written in lowercase, with the exception of parameter values, whose case may or may not have specific meaning.

Typ

There are two classes of type: **discrete** and **multipart**. Discrete types are types which represent a single file or medium, such as a single text or music file, or a single video. A multipart type is one which represents a document that's comprised of multiple component parts, each of which may have its own individual MIME type; or, a multipart type may encapsulate multiple files being sent together in one transaction. For example, multipart MIME types are used when attaching multiple files to an email.

Discrete types

The discrete types currently registered with the IANA are:

applicationList at IANA

> Any kind of binary data that doesn't fall explicitly into one of the other types; either data that will be executed or interpreted in some way or binary data that requires a specific application or category of application to use. Generic binary data (or binary data whose true type is unknown) is application/octet-stream. Other common examples include application/pdf, application/pkcs8, and application/zip.

audioList at IANA

> Audio or music data. Examples include audio/mpeg, audio/vorbis.

example

> Reserved for use as a placeholder in examples showing how to use MIME types. These should never be used outside of sample code listings and documentation. example can also be used as a subtype; for instance, in an example related to working with audio on the web, the MIME type audio/example can be used to indicate that the type is a

placeholder and should be replaced with an appropriate one when using the code in the real world.

fontList at IANA

Font/typeface data. Common examples include font/woff, font/ttf, and font/otf.

imageList at IANA

Image or graphical data including both bitmap and vector still images as well as animated versions of still image formats such as animated GIF or APNG. Common examples are image/jpeg, image/png, and image/svg+xml.

modelList at IANA

Model data for a 3D object or scene. Examples include model/3mf and model/vml.

textList at IANA

Text-only data including any human-readable content, source code, or textual data such as comma-separated value (CSV) formatted data. Examples include text/plain, text/csv, and text/html.

videoList at IANA

Video data or files, such as MP4 movies (video/mp4).

For text documents without a specific subtype, text/plain should be used. Similarly, for binary documents without a specific or known subtype, application/octet-stream should be used.

Multipart types

Multipart types indicate a category of document broken into pieces, often with different MIME types; they can also be used — especially in email scenarios — to represent multiple, separate files which are all part of the same transaction. They represent a **composite document**.
With the exception of multipart/form-data, used in the POST method of HTML Forms, and multipart/byteranges, used with 206 Partial Content to send part of a document, HTTP doesn't handle multipart documents in a special way: the message is transmitted to the browser (which will likely show a "Save As" window if it doesn't know how to display the document). There are two multipart types:

messageList at IANA

A message that encapsulates other messages. This can be used, for instance, to represent an email that includes a forwarded message as part of its data, or to allow sending very large messages in chunks as if it were multiple messages. Examples

include message/rfc822 (for forwarded or replied-to message quoting)
and message/partial to allow breaking a large message into smaller ones automatically to be reassembled by the recipient.

multipartList at IANA

Data that is comprised of multiple components which may individually have different MIME types. Examples include multipart/form-data (for data produced using the FormData API) and multipart/byteranges (defined in RFC 7233: 5.4.1 and used with HTTP's 206 "Partial Content" response returned when the fetched data is only part of the content, such as is delivered using the Range header).

Working with Plug-Ins and Helper Applications

Many file types are available on the Internet. In addition to HTML, JSP, GIF, JPEG, and other files that are used to present a Web page, there are graphics, movies, sounds, and many other file types you can open and view. Internet Explorer can't work with all these file types directly, and fortunately, it doesn't have to. Internet Explorer and other Web browsers use plug-ins and helper applications to expand their capabilities so that they can work with files that they don't natively support.

Working with Plug-Ins

Plug-ins are software that can be incorporated into a Web browser when it opens (thus, the term plug-in). Internet plug-ins enable applications to display files that are of the specific types handled by those plug-ins. For example, the QuickTime plug-in enables Web browsers to display QuickTime movies.

Installing Internet Plug-Ins

As you travel around the Web, you might encounter file types for which you do not have the required plug-in. In that case, you need to find and install the plug-in you need. Usually, sites will have links to places from which you can download the plug-ins needed for the file types on the site. There are a couple of places in the system where plug-ins can be stored.

Plug-ins that are available to all user accounts are stored in the folder Mac OS X/Library/Internet Plug-Ins/, where Mac OS X is the name of your startup volume.

You must be logged in under an Administrator account to be able to store a plug-in in this directory.

Internet plug-ins can also be stored in a specific user account, in which case they are available only to that user. A user's specific plug-ins are in the location shortusername/Library/Internet Plug-Ins/, where shortusername is the short name for the user account.

To install a plug-in, simply place it in the directory that is appropriate for that plug-in (to be available either to all users or to only a specific user). Quit the Web browser and then launch it again to make the plug-in active.

NOTE

Some plug-ins are installed using an installer application, in which case you don't need to install the plug-in manually.

If you open the Internet Plug-Ins directories, you will see the plug-ins that are currently installed. Any plug-in installed in this folder can be used by a supported Web browser.

Many plug-ins are available for Web browsers. The QuickTime plug-in is installed by default so that you can view QuickTime movies in Web browsers. Additionally, the Shockwave Flash plug-in is installed by default as is the Java Applet plug-in. There are many other plug-ins you might want to download and install.

When you attempt to view a file for which you do not have the appropriate plug-in, you will see a warning dialog box that tells you what to do. Usually, you see instructions to help you find, download, and install the plug-in as well.

Using Internet Plug-Ins

After a plug-in is installed in the appropriate folder, it works with a Web browser to provide its capabilities. When you click a file that requires the plug-in to be used, the appropriate plug-in activates and enables you to do whatever it is designed to do. For example, when you open a QuickTime movie, you see the controls that enable you to watch that movie within the Web browser.

Working with Helper Applications

Although plug-ins provide additional capability by "plugging in" to a Web browser, helper applications are standalone applications that Web browsers can use to work with files of specific types. Any application on your Mac can be used as a helper application.

Web browsers maintain a list of document types for which helper applications should be used. When you open a file type that has a helper application, that application opens and you can use it to work with a file. Web browsers have a default helper application list that links helper applications to many different document types. You can add, change, or delete file types and applications from this list.

For example, in the previous section, you learned how StuffIt Expander can decode and uncompress files you download from the Internet. StuffIt Expander is a helper application; as you saw earlier, you can also use the application independently of a browser. When a browser opens a file type for which StuffIt Expander is the designated helper application, the browser opens StuffIt Expander and "passes" the file to it for processing.

Using Helper Applications

Using helper applications is just like using those applications as standalone applications (after the browser opens the application and passes a file to it). For example, suppose that the application Microsoft Word is the designated helper application for files with the .doc file extension. When you click a file whose name ends in .doc, the browser launches Word and passes the document file to it. The document opens in Word and you can work with it just as you can any other Word document.

Configuring Helper Applications in Internet Explorer

Using helper applications is simple, but determining which helper applications are used can be a bit more difficult. You need to be able to relate specific file extensions to the application you want to use as the helper application for that file type.

The best way to see how this works is to work through an example. In this example, the application StuffIt Expander will be designated as the helper application for Zip files you download. This means that when you download a Zip file, StuffIt Expander will be opened and will unzip the file automatically.

NOTE

By default, Internet Explorer already uses StuffIt Expander as the file helper associated with Zip files. However, this example provides steps that are typical when you need to associate a file with a file helper and so is a worthwhile exercise.

1. Open the Internet Explorer Preferences window and click File Helpers to open the File Helpers Settings pane. This pane provides a list of applications along with their descriptions. Next to each, you will see the filename extension and MIME file type of the files for which that application is the helper.
2. Click Add and the Edit File Helper window opens. This window provides the controls and fields you need to designate a helper application for any file type. The window will be empty when you create an association.
3. Describe the association you are creating.

   In the example, I used "Zip files."
4. Press Tab and enter the extension for the file type for which you are creating an association.

   In this example, I entered ".zip" as the extension.
5. In the MIME type field, enter application/zip.
6. Enter the four-letter file type and creator codes for the files you will be associating with this application.

   In the example, I entered "ZIP " in both places (the letters "ZIP" followed by a space).

   NOTE

   Explaining MIME, file types, and file creators is beyond what I have room to cover here. Usually, you can find what you need to enter by looking at one of the existing associations. In fact, that will often work for any new association you want to create. Use the existing associations as guides.
7. Using the How to Handle pop-up menu, choose how you want these files to be handled.

   Because I want the files to be processed with the application (rather than being viewed, for example), I chose Post-Process with Application.

8. Use the Open dialog box to choose the application that you want to be the helper application for files of this type.

   In the example, I selected StuffIt Expander.

9. Click OK to return to the File Helper Settings pane and you will see your new association in the window.

XML

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions −

- **XML is extensible** − XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it** − XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard** − XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage

A short list of XML usage says it all −

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. So what exactly is a markup language? Markup is

information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text −

```
<message>
  <text>Hello, world!</text>
</message>
```

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>... </text>. The tags <message> and </message> mark the start and the end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!.

Is XML a Programming Language?

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instruct the computer to perform specific tasks. XML does not qualify to be a programming language as it does not perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

XHTML - Introduction

XHTML stands for E**X**tensible **H**yper**T**ext **M**arkup **L**anguage. It is the next step in the evolution of the internet. The XHTML 1.0 is the first document type in the XHTML family.

XHTML is almost identical to HTML 4.01 with only few differences. This is a cleaner and stricter version of HTML 4.01. If you already know HTML, then you need to give little attention to learn this latest version of HTML.

XHTML was developed by World Wide Web Consortium (W3C) to help web developers make the transition from HTML to XML. By migrating to XHTML today, web developers can enter the XML world with all of its benefits, while still remaining confident in the backward and future compatibility of the content.

Why Use XHTML?

Developers who migrate their content to XHTML 1.0 get the following benefits −

- XHTML documents are XML conforming as they are readily viewed, edited, and validated with standard XML tools.
- XHTML documents can be written to operate better than they did before in existing browsers as well as in new browsers.
- XHTML documents can utilize applications such as scripts and applets that rely upon either the HTML Document Object Model or the XML Document Object Model.

- XHTML gives you a more consistent, well-structured format so that your webpages can be easily parsed and processed by present and future web browsers.
- You can easily maintain, edit, convert and format your document in the long run.
- Since XHTML is an official standard of the W3C, your website becomes more compatible with many browsers and it is rendered more accurately.
- XHTML combines strength of HTML and XML. Also, XHTML pages can be rendered by all XML enabled browsers.
- XHTML defines quality standard for your webpages and if you follow that, then your web pages are counted as quality web pages. The W3C certifies those pages with their quality stamp.

Web developers and web browser designers are constantly discovering new ways to express their ideas through new markup languages. In XML, it is relatively easy to introduce new elements or additional element attributes. The XHTML family is designed to accommodate these extensions through XHTML modules and techniques for developing new XHTML-conforming modules. These modules permit the combination of existing and new features at the time of developing content and designing new user agents.

Basic Understanding

Before we proceed further, let us have a quick view on what are HTML, XML, and SGML.

What is SGML?

This is **S**tandard **G**eneralized **M**arkup **L**anguage (SGML) application conforming to International Standard ISO 8879. HTML is widely regarded as the standard publishing language of the World Wide Web.

This is a language for describing markup languages, particularly those used in electronic document exchange, document management, and document publishing. HTML is an example of a language defined in SGML.

What is XML?

XML stands for E**X**tensible **M**arkup **L**anguage. XML is a markup language much like HTML and it was designed to describe data. XML tags are not predefined. You must define your own tags according to your needs.

XHTML - Syntax

XHTML syntax is very similar to HTML syntax and almost all the valid HTML elements are valid in XHTML as well. But when you write an XHTML document, you need to pay a bit extra attention to make your HTML document compliant to XHTML.

Here are the important points to remember while writing a new XHTML document or converting existing HTML document into XHTML document −

- Write a DOCTYPE declaration at the start of the XHTML document.

- Write all XHTML tags and attributes in lower case only.
- Close all XHTML tags properly.
- Nest all the tags properly.
- Quote all the attribute values.
- Forbid Attribute minimization.
- Replace the **name** attribute with the **id** attribute.
- Deprecate the **language** attribute of the script tag.

Here is the detail explanation of the above XHTML rules −

DOCTYPE Declaration

All XHTML documents must have a DOCTYPE declaration at the start. There are three types of DOCTYPE declarations, which are discussed in detail in XHTML Doctypes chapter. Here is an example of using DOCTYPE −

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

Case Sensitivity

XHTML is case sensitive markup language. All the XHTML tags and attributes need to be written in lower case only.

```
<!-- This is invalid in XHTML -->
<A Href="/xhtml/xhtml_tutorial.html">XHTML Tutorial</A>

<!-- Correct XHTML way of writing this is as follows -->
<a href="/xhtml/xhtml_tutorial.html">XHTML Tutorial</a>
```

In the example, **Href** and anchor tag **A** are not in lower case, so it is incorrect.

Closing the Tags

Each and every XHTML tag should have an equivalent closing tag, even empty elements should also have closing tags. Here is an example showing valid and invalid ways of using tags −

```
<!-- This is invalid in XHTML -->
<p>This paragraph is not written according to XHTML syntax.

<!-- This is also invalid in XHTML -->
<img src="/images/xhtml.gif" >
```

The following syntax shows the correct way of writing above tags in XHTML. Difference is that, here we have closed both the tags properly.

```
<!-- This is valid in XHTML -->
<p>This paragraph is not written according to XHTML syntax.</p>

<!-- This is also valid now -->
<img src="/images/xhtml.gif" />
```

Attribute Quotes

All the values of XHTML attributes must be quoted. Otherwise, your XHTML document is assumed as an invalid document. Here is the example showing syntax −

```
<!-- This is invalid in XHTML -->
<img src="/images/xhtml.gif" width=250 height=50 />

<!-- Correct XHTML way of writing this is as follows -->
<img src="/images/xhtml.gif" width="250" height="50" />
```

Attribute Minimization

XHTML does not allow attribute minimization. It means you need to explicitly state the attribute and its value. The following example shows the difference −

```
<!-- This is invalid in XHTML -->
<option selected>

<!-- Correct XHTML way of writing this is as follows -->
<option selected="selected">
```

Here is a list of the minimized attributes in HTML and the way you need to write them in XHTML −

HTML Style	XHTML Style
Compact	compact="compact"
Checked	checked="checked"
Declare	declare="declare"
Readonly	readonly="readonly"
Disabled	disabled="disabled"

Selected	selected="selected"
Defer	defer="defer"
Ismap	ismap="ismap"
Nohref	nohref="nohref"
Noshade	noshade="noshade"
Nowrap	nowrap="nowrap"
Multiple	multiple="multiple"
Noresize	noresize="noresize"

The id Attribute

The id attribute replaces the name attribute. Instead of using name = "name", XHTML prefers to use id = "id". The following example shows how −

```
<!-- This is invalid in XHTML -->
<img src="/images/xhtml.gif" name="xhtml_logo" />

<!-- Correct XHTML way of writing this is as follows -->
<img src="/images/xhtml.gif" id="xhtml_logo" />
```

The language Attribute

The language attribute of the script tag is deprecated. The following example shows this difference −

```
<!-- This is invalid in XHTML -->

<script language="JavaScript" type="text/JavaScript">
  document.write("Hello XHTML!");
</script>

<!-- Correct XHTML way of writing this is as follows -->
```

```
<script type="text/JavaScript">
   document.write("Hello XHTML!");
</script>
```

Nested Tags

You must nest all the XHTML tags properly. Otherwise your document is assumed as an incorrect XHTML document. The following example shows the syntax −

```
<!-- This is invalid in XHTML -->
<b><i> This text is bold and italic</b></i>

<!-- Correct XHTML way of writing this is as follows -->
<b><i> This text is bold and italic</i></b>
```

Element Prohibitions

The following elements are not allowed to have any other element inside them. This prohibition applies to all depths of nesting. Means, it includes all the descending elements.

Element	Prohibition
<a>	Must not contain other <a> elements.
<pre>	Must not contain the <img>, <object>, <big>, <small>, <sub>, or <sup> elements.
<button>	Must not contain the <input>, <select>, <textarea>, <label>, <button>, <form>, <fieldset>, <iframe> or <isindex> elements.
<label>	Must not contain other <label> elements.
<form>	Must not contain other <form> elements.

A Minimal XHTML Document

The following example shows you a minimum content of an XHTML 1.0 document −

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

```
<!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/TR/xhtml1" xml:lang="en" lang="en">
  <head>
    <title>Every document must have a title</title>
  </head>

  <body>
    ...your content goes here...
  </body>
</html>
```

HTML Versus XHTML

Due to the fact that XHTML is an XML application, certain practices that were perfectly legal in SGML-based HTML 4 must be changed. You already have seen XHTML syntax in previous chapter, so differences between XHTML and HTML are very obvious. Following is the comparison between XHTML and HTML.

XHTML Documents Must be Well-Formed

Well-formedness is a new concept introduced by XML. Essentially, this means all the elements must have closing tags and you must nest them properly.

CORRECT: Nested Elements

```
<p>Here is an emphasized <em>paragraph</em>.</p>
```

INCORRECT: Overlapping Elements

```
<p>Here is an emphasized <em>paragraph.</p></em>
```

Elements and Attributes Must be in Lower Case

XHTML documents must use lower case for all HTML elements and attribute names. This difference is necessary because XHTML document is assumed to be an XML document and XML is case-sensitive. For example, <li> and <LI> are different tags.

End Tags are Required for all Elements

In HTML, certain elements are permitted to omit the end tag. But XML does not allow end tags to be omitted.

CORRECT: Terminated Elements

```
<p>Here is a paragraph.</p><p>here is another paragraph.</p>
<br><hr/>
```

INCORRECT: Unterminated Elements

```
<p>Here is a paragraph.<p>here is another paragraph.
<br><hr>
```

Attribute Values Must Always be Quoted

All attribute values including numeric values, must be quoted.

CORRECT: Quoted Attribute Values

```
<td rowspan="3">
```

INCORRECT: Unquoted Attribute Values

```
<td rowspan=3>
```

Attribute Minimization

XML does not support attribute minimization. Attribute-value pairs must be written in full. Attribute names such as compact and checked cannot occur in elements without their value being specified.

CORRECT: Non Minimized Attributes

```
<dl compact="compact">
```

INCORRECT: Minimized Attributes

```
<dl compact>
```

Whitespace Handling in Attribute Values

When a browser processes attributes, it does the following −

- Strips leading and trailing whitespace.
- Maps sequences of one or more white space characters (including line breaks) to a single inter-word space.

Script and Style Elements

In XHTML, the script and style elements should not have “<” and “&” characters directly, if they exist; then they are treated as the start of markup. The entities such as “<” and “&” are recognized as entity references by the XML processor for displaying “<” and “&” characters respectively.

Wrapping the content of the script or style element within a CDATA marked section avoids the expansion of these entities.

```
<script type="text/JavaScript">
  <![CDATA[
    ... unescaped VB or Java Script here... ...
```

```
   ]]>
</script>
```

An alternative is to use external script and style documents.

The Elements with id and name Attributes

XHTML recommends the replacement of name attribute with id attribute. Note that in XHTML 1.0, the name attribute of these elements is formally deprecated, and it will be removed in a subsequent versions of XHTML.

Attributes with Pre-defined Value Sets

HTML and XHTML both have some attributes that have pre-defined and limited sets of values. For example, **type** attribute of the **input** element. In HTML and XML, these are called **enumerated attributes**. Under HTML 4, the interpretation of these values was case-insensitive, so a value of **TEXT** was equivalent to a value of **text**.

Under XHTML, the interpretation of these values is case-sensitive so all of these values are defined in lower-case.

Entity References as Hex Values

HTML and XML both permit references to characters by using hexadecimal value. In HTML these references could be made using either **&#Xnn;** or **&#xnn;** and they are valid but in XHTML documents, you must use the lower-case version only such as **&#xnn;**.

The <html> Element is a Must

All XHTML elements must be nested within the <html> root element. All other elements can have sub elements which must be in pairs and correctly nested within their parent element. The basic document structure is −

```
<!DOCTYPE html >

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

XHTML - Doctypes

The XHTML standard defines three Document Type Definitions (DTDs). The most commonly used and easy one is the XHTML Transitional document.

XHTML 1.0 document type definitions correspond to three DTDs −

- Strict
- Transitional

- Frameset

There are few XHTML elements and attributes, which are available in one DTD but not available in another DTD. Therefore, while writing your XHTML document, you must select your XHTML elements or attributes carefully. However, XHTML validator helps you to identify valid and invalid elements and attributes.

Please check XHTML Validations for more detail on this.

XHTML 1.0 Strict

If you are planning to use Cascading Style Sheet (CSS) strictly and avoiding to write most of the XHTML attributes, then it is recommended to use this DTD. A document conforming to this DTD is of the best quality.

If you want to use XHTML 1.0 Strict DTD then you need to include the following line at the top of your XHTML document.

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

XHTML 1.0 Transitional

If you are planning to use many XHTML attributes as well as few Cascading Style Sheet properties, then you should adopt this DTD and you should write your XHTML document accordingly.

If you want to use XHTML 1.0 Transitional DTD, then you need to include the following line at the top of your XHTML document.

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

XHTML 1.0 Frameset

You can use this when you want to use HTML Frames to partition the browser window into two or more frames.

If you want to use XHTML 1.0 Frameset DTD, then you need to include following line at the top of your XHTML document.

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

Note − No matter what DTD you are using to write your XHTML document; if it is a valid XHTML document, then your document is considered as a good quality document.

XHTML - Attributes

There are a few XHTML/HTML attributes which are standard and associated to all the XHTML/HTML tags. These attributes are listed here with brief description −

Core Attributes

Not valid in base, head, html, meta, param, script, style, and title elements.

Attribute	Value	Description
Class	class_rule or style_rule	The class of the element.
Id	id_name	A unique id for the element.
Style	style_definition	An inline style definition.
Title	tooltip_text	A text to display in a mouse tip.

Language Attributes

The **lang** attribute indicates the language being used for the enclosed content. The language is identified using the ISO standard language abbreviations, such as **fr** for French, **en** for English, and so on. More codes and their formats are described at www.ietf.org.

Not valid in base, br, frame, frameset, hr, iframe, param, and script elements.

Attribute	Value	Description
Dir	ltr | rtl	Sets the text direction.
Lang	language_code	Sets the language code.

Microsoft Proprietary Attributes

Microsoft introduced a number of new proprietary attributes with the Internet Explorer 4 and higher versions.

Attribute	Value	Description
accesskey	Character	Sets a keyboard shortcut to access an element.

language	String	This attribute specifies the scripting language to be used with an associated script bound to the element, typically through an event handler attribute. Possible values might include JavaScript, jScript, VBS, and VBScript.
tabindex	Number	Sets the tab order of an element.
contenteditable	Boolean	Allows users to edit content rendered in Internet Explorer 5.5 or greater. Possible values are true or false.
disabled	Boolean	Elements with the disabled attribute set may appear faded and will not respond to user input. Possible values are true or false.
hidefocus	on or off	This proprietary attribute, introduced with Internet Explorer 5.5, hides focus on an element's content. Focus must be applied to the element using the tabindex attribute.
unselectable	on or off	Used to prevent content displayed in Internet Explorer 5.5 from being selected.

XHTML - Events

When users visit a website, they do things such as click on text, images and hyperlinks, hover-over things, etc. These are examples of what JavaScript calls events.

We can write our event handlers in JavaScript or VBScript and can specify these event handlers as a value of event tag attribute. The XHTML 1.0 has a similar set of events which is available in HTML 4.01 specification.

The <body> and <frameset> Level Events

There are only two attributes which can be used to trigger any JavaScript or VBScript code, when any event occurs at document level.

Attribute	Value	Description

Onload	Script	Script runs when a XHTML document loads.
onunload	Script	Script runs when a XHTML document unloads.

Note − Here, the script refers to any function or piece of code of VBScript or JavaScript.

The <form> Level Events

There are following six attributes which can be used to trigger any JavaScript or VBScript code when any event occurs at form level.

Attribute	Value	Description
onchange	Script	Script executes when the element changes.
onsubmit	Script	Script executes when the form is submitted.
Onreset	Script	Script executes when the form is reset.
onselect	Script	Script executes when the element is selected.
Onblur	Script	Script executes when the element loses focus.
onfocus	Script	Script runs when the element gets focus.

Keyboard Events

The following three events are generated by keyboard. These events are not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, and title elements.

Attribute	Value	Description
onkeydown	Script	Script executes on key press.
onkeypress	Script	Script executes on key press and release.

onkeyup	Script	Script executes key release.

Other Events

The following seven events are generated by mouse when it comes in contact with any HTML tag. These events are not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, and title elements.

Attribute	Value	Description
Onclick	Script	Script executes on a mouse click.
ondblclick	Script	Script executes on a mouse double-click.
onmousedown	Script	Script executes when mouse button is pressed.
onmousemove	Script	Script executes when mouse pointer moves.
onmouseout	Script	Script executes when mouse pointer moves out of an element.
onmouseover	Script	Script executes when mouse pointer moves over an element.
onmouseup	Script	Script executes when mouse button is released.

XHTML - Version 1.1

The W3C has helped move the internet content-development community from the days of malformed, non-standard mark-up into the well-formed, valid world of XML. In XHTML 1.0, this move was moderated by the goal of providing easy migration of existing HTML 4 (or earlier) based content to XHTML and XML.

The W3C has removed support for deprecated elements and attributes from the XHTML family. These elements and attributes had largely presentation-oriented functionality that is better handled via style sheets or client-specific default behavior.

Now the W3C's HTML Working Group has defined an initial document type based solely upon modules which are XHTML 1.1. This document type is designed to be portable to a broad collection of client devices, and applicable to the majority of internet content.

Document Conformance

The XHTML 1.1 provides a definition of strictly conforming XHTML documents which MUST meet all the following criteria −

- The document MUST conform to the constraints expressed in XHTML 1.1 Document Type Definition.
- The root element of the document MUST be <html>.
- The root element of the document MUST designate the XHTML namespace using the xmlns attribute.
- The root element MAY also contain a schema location attribute as defined in the XML Schema.

There MUST be a DOCTYPE declaration in the document prior to the root element. If it is present, the public identifier included in the DOCTYPE declaration MUST refer the DTD found in XHTML 1.1 Document Type Definition.

Here is an example of an XHTML 1.1 document −

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/MarkUp/SCHEMA/xhtml11.xsd" xml:lang="en">

   <head>
      <title>This is the document title</title>
   </head>

   <body>
      <p>Moved to <a href="http://example.org/">example.org</a>.</p>
   </body>

</html>
```

Note − In this example, the XML declaration is included. An XML declaration such as the one above is not required in all XML documents. XHTML document authors are strongly encouraged to use XML declarations in all their documents. Such a declaration is required when the character encoding of the document is other than the default UTF-8 or UTF-16.

XHTML 1.1 Modules

The XHTML 1.1 document type is made up of the following XHTML modules.

Structure Module − The Structure Module defines the major structural elements for XHTML. These elements effectively act as the basis for the content model of many XHTML family document types. The elements and attributes included in this module are − body, head, html, and title.

Text Module − This module defines all of the basic text container elements, attributes, and their content model − abbr, acronym, address, blockquote, br, cite, code, dfn, div, em, h1, h2, h3, h4, h5, h6, kbd, p, pre, q, samp, span, strong, and var.

Hypertext Module − The Hypertext Module provides the element that is used to define hypertext links to other resources. This module supports element a.

List Module − As its name suggests, the List Module provides list-oriented elements. Specifically, the List Module supports the following elements and attributes − dl, dt, dd, ol, ul, and li.

Object Module − The Object Module provides elements for general-purpose object inclusion. Specifically, the Object Module supports − object and param.

Presentation Module − This module defines elements, attributes, and a minimal content model for simple presentation-related markup − b, big, hr, i, small, sub, sup, and tt.

Edit Module − This module defines elements and attributes for use in editing-related markup − del and ins.

Bidirectional Text Module − The Bi-directional Text module defines an element that can be used to declare the bi-directional rules for the element's content − bdo.

Forms Module − It provides all the form features found in HTML 4.0. Specifically, it supports − button, fieldset, form, input, label, legend, select, optgroup, option, and textarea.

Table Module − It supports the following elements, attributes, and content model − caption, col, colgroup, table, tbody, td, tfoot, th, thead, and tr.

Image Module − It provides basic image embedding and may be used in some implementations of client side image maps independently. It supports the element − img.

Client-side Image Map Module − It provides elements for client side image maps − area and map.

Server-side Image Map Module − It provides support for image-selection and transmission of selection coordinates. The Server-side Image Map Module supports − attribute ismap on img.

Intrinsic Events Module − It supports all the events discussed in XHTML Events.

Meta information Module − The Meta information Module defines an element that describes information within the declarative portion of a document. It includes element meta.

Scripting Module − It defines the elements used to contain information pertaining to executable scripts or the lack of support for executable scripts. Elements and attributes included in this module are − noscript and script.

Style Sheet Module − It defines an element to be used when declaring internal style sheets. The element and attribute defined by this module is − style.

Style Attribute Module (Deprecated) − It defines the style attribute.

Link Module − It defines an element that can be used to define links to external resources. It supports link element.

Base Module − It defines an element that can be used to define a base URI against which relative URIs in the document are resolved. The element and attribute included in this module is − base.

Ruby Annotation Module − XHTML also uses the Ruby Annotation module as defined in RUBY and supports − ruby, rbc, rtc, rb, rt, and rp.

Changes from XHTML 1.0 Strict

This section describes the differences between XHTML 1.1 and XHTML 1.0 Strict. XHTML 1.1 represents a departure from both HTML 4 and XHTML 1.0.

- The most significant is the removal of features that were deprecated.
- The changes can be summarized as follows −
- On every element, the lang attribute has been removed in favor of the xml:lang attribute.
- On the <a> and <map> elements, the name attribute has been removed in favor of the id attribute.
- The ruby collection of elements has been added.

XHTML - Tips & Tricks

This chapter lists out various tips and tricks which you should be aware of while writing an XHTML document. These tips and tricks can help you create effective documents.

Tips for Designing XHTML Document

Here are some basic guidelines for designing XHTML documents −

Design for Serving and Engaging Your Audience

When you think of satisfying what your audience wants, you need to design effective and catchy documents to serve the purpose. Your document should be easy for finding required information and giving a familiar environment.

For example, Academicians or medical practitioners are comfortable with journal-like document with long sentences, complex diagrams, specific terminologies, etc., whereas the document accessed by school-going children must be simple and informative.

Reuse Your Document

Reuse your previously created successful documents instead of starting from scratch each time you bag a new project.

Inside the XHTML Document

Here are some tips regarding elements inside the XHTML document −

The XML Declaration

An XML declaration is not required in all XHTML documents but XHTML document authors are strongly encouraged to use XML declarations in all their documents. Such a declaration is required when the character encoding of the document is other than the default UTF-8 or UTF-16.

Empty Elements

They include a space before the trailing / and > of empty elements. For example,
, <hr />, and .

Embedded Style Sheets and Scripts

Use external style sheets if your style sheet uses “<”, “&”, “]]>”, or “—”.

Use external scripts if your script uses “<”, “&”, or “]]>”, or “—”.

Line Breaks within Attribute Values

Avoid line breaks and multiple whitespace characters within attribute values. These are handled inconsistently by different browsers.

Isindex Element

Do not include more than one isindex element in the document head. The isindex element is deprecated in favor of the input element.

The lang and xml:lang Attributes

Use both the lang and xml:lang attributes while specifying the language of an element. The value of the xml:lang attribute takes precedence.

Element Identifiers

XHTML 1.0 has deprecated the name attributes of a, applet, form, frame, iframe, img, and map elements. They will be removed from XHTML in subsequent versions. Therefore, start using id element for element identification.

Using Ampersands in Attribute Values

The ampersand character ("&") should be presented as an entity reference &.

Example

```
<!-- This is invalid in XHTML -->
http://my.site.dom/cgi-bin/myscript.pl?class=guest&name=user.

<!-- Correct XHTML way of writing this is as follows -->
http://my.site.dom/cgi-bin/myscript.pl?class=guest&name=user
```

Whitespace Characters in HTML and XML

Some characters that are legal in HTML documents are illegal in XML document. For example, in HTML, the form-feed character (U+000C) is treated as white space, in XHTML, due to XML's definition of characters, it is illegal.

Named Character Reference &Apos;

The named character reference ' (the apostrophe, U+0027) was introduced in XML 1.0 but does not appear in HTML. Web developers should therefore use ' instead of ' to work as expected in HTML 4 Web Browsers.

XHTML - Validations

Every XHTML document is validated against a Document Type Definition. Before validating an XHTML file properly, a correct DTD must be added as the first or second line of the file.

Once you are ready to validate your XHTML document, you can use W3C Validator to validate your document. This tool is very handy and helps you to fix the problems with your document. This tool does not require any expertise to perform validation.

The following statement in the text box shows you details. You need to give complete URL of the page, which you want to validate and then click **Validate Page** button.

Input your page address in the box below −

This validator checks the markup validity of web documents with various formats especially in HTML, XHTML, SMIL, MathML, etc.

There are other tools to perform different other validations.

- RSS/Atom feeds Validator
- CSS stylesheets Validator
- Find Broken Links
- Other validators and tools

W3C

The W3C DOM standardizes most of the features of the legacy DOM and adds new ones as well. In addition to supporting forms[ ], images[ ], and other array properties of the Document object, it defines methods that allow scripts to access and manipulate any document element and not just special-purpose elements like forms and images.

Document Properties in W3C DOM

This model supports all the properties available in Legacy DOM. Additionally, here is a list of document properties which can be accessed using W3C DOM.

Sr.No.	Property & Description
1	**body** A reference to the Element object that represents the <body> tag of this document. **Ex** − document.body
2	**defaultView** Its Read-only property and represents the window in which the document is displayed. **Ex** − document.defaultView
3	**documentElement** A read-only reference to the <html> tag of the document. **Ex** − document.documentElement8/31/2008
4	**implementation** It is a read-only property and represents the DOMImplementation object that represents the implementation that created this document. **Ex** − document.implementation

Document Methods in W3C DOM

This model supports all the methods available in Legacy DOM. Additionally, here is a list of methods supported by W3C DOM.

Sr.No.	Property & Description
1	**createAttribute(name)** Returns a newly-created Attr node with the specified name. **Ex** − document.createAttribute(name)

2	**createComment(text)** Creates and returns a new Comment node containing the specified text. **Ex** − document.createComment(text)
3	**createDocumentFragment( )** Creates and returns an empty DocumentFragment node. **Ex** − document.createDocumentFragment( )
4	**createElement(tagName)** Creates and returns a new Element node with the specified tag name. **Ex** − document.createElement(tagName)
5	**createTextNode(text)** Creates and returns a new Text node that contains the specified text. **Ex** − document.createTextNode(text)
6	**getElementById(id)** Returns the Element of this document that has the specified value for its id attribute, or null if no such Element exists in the document. **Ex** − document.getElementById(id)
7	**getElementsByName(name)** Returns an array of nodes of all elements in the document that have a specified value for their name attribute. If no such elements are found, returns a zero-length array. **Ex** − document.getElementsByName(name)
8	**getElementsByTagName(tagname)** Returns an array of all Element nodes in this document that have the specified tag name. The Element nodes appear in the returned array in the same order they appear in the document source. **Ex** − document.getElementsByTagName(tagname)

9	**importNode(importedNode, deep)** Creates and returns a copy of a node from some other document that is suitable for insertion into this document. If the deep argument is true, it recursively copies the children of the node too. Supported in DOM Version 2 **Ex** − document.importNode(importedNode, deep)

Example

This is very easy to manipulate (Accessing and Setting) document element using W3C DOM. You can use any of the methods like **getElementById**, **getElementsByName**, or **getElementsByTagName**.

Here is an example to access document properties using W3C DOM method.

Live Demo

```
<html>
  <head>
    <title> Document Title </title>
    <script type = "text/javascript">
      <!--
        function myFunc() {
          var ret = document.getElementsByTagName("title");
          alert("Document Title : " + ret[0].text );

          var ret = document.getElementById("heading");
          alert(ret.innerHTML );
        }
      //-->
    </script>
  </head>
  <body>
    <h1 id = "heading">This is main title</h1>
    <p>Click the following to see the result:</p>

    <form id = "form1" name = "FirstForm">
      <input type = "button" value = "Click Me" onclick = "myFunc();" />
      <input type = "button" value = "Cancel">
    </form>

    <form d = "form2" name = "SecondForm">
      <input type = "button" value = "Don't ClickMe"/>
    </form>
  </body>
```

```
</html>
```

Anatomy of HTML document

HTML stands for **Hyper Text Markup Language.** It is a formatting language used to define the appearance and contents of a web page. It allows us to organize text, graphics, audio, and video on a web page.

Key Points:

- The word Hypertext refers to the text which acts as a link.
- The word markup refers to the symbols that are used to define structure of the text. The markup symbols tells the browser how to display the text and are often called tags.
- The word Language refers to the syntax that is similar to any other language.

HTML was created by **Tim Berners-Lee** at **CERN.**

HTML Versions

The following table shows the various versions of HTML:

Version	Year
HTML 1.0	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.0	1999
XHTML	2000

HTML5	2012

HTML Tags

Tag is a command that tells the web browser how to display the text, audio, graphics or video on a web page.

Key Points:

- Tags are indicated with pair of angle brackets.
- They start with a less than (<) character and end with a greater than (>) character.
- The tag name is specified between the angle brackets.
- Most of the tags usually occur in pair: the start tag and the closing tag.
- The start tag is simply the tag name is enclosed in angle bracket whereas the closing tag is specified including a forward slash (/).
- Some tags are the empty i.e. they don't have the closing tag.
- Tags are not case sensitive.
- The starting and closing tag name must be the same. For example <b> hello </i> is invalid as both are different.
- If you don't specify the angle brackets (<>) for a tag, the browser will treat the tag name as a simple text.
- The tag can also have attributes to provide additional information about the tag to the browser.

Basic tags

The following table shows the Basic HTML tags that define the basic web page:

Tag	Description
<html> </html>	Specifies the document as a web page.
<head> </head>	Specifies the descriptive information about the web documents.
<title> </title>	Specifies the title of the web page.

<body> </body>	Specifies the body of a web document.

The following code shows how to use basic tags.

```
<html>
  <head> Heading goes here…</head>
  <title> Title goes here…</title>
  <body> Body goes here…</body>
</html>
```

Formatting Tags

The following table shows the HTML tags used for formatting the text:

Tag	Description
	Specifies the text as bold. Eg. **this is bold text**
	It is a phrase text. It specifies the emphasized text. Eg. *Emphasized text*
	It is a phrase tag. It specifies an important text. Eg. **this is strong text**
	The content of italic tag is displayed in italic. Eg. Italic text
	Specifies the subscripted text. Eg. X_1
	Defines the superscripted text. Eg. X^2
<ins> </ins>	Specifies the inserted text. Eg. The price of pen is now 15.
<del> </del>	Specifies the deleted text. Eg. The price of pen is now 15.
<mark> </mark>	Specifies the marked text. Eg. It is raining

Table Tags

Following table describe the commonaly used table tags:

Tag	Description
<table> </table>	Specifies a table.
<tr> </tr>	Specifies a row in the table.
<th> </th>	Specifies header cell in the table.
<td> </td>	Specifies the data in an cell of the table.
<caption> </caption>	Specifies the table caption.
<colgroup> </colgroup>	Specifies a group of columns in a table for formatting.

List tags

Following table describe the commonaly used list tags:

Tag	Description
	Specifies an unordered list.
	Specifies an ordered list.
	Specifies a list item.
<dl> </dl>	Specifies a description list.
<dt> </dt>	Specifies the term in a description list.

<dd> </dd>	Specifies description of term in a description list.

Frames

Frames help us to divide the browser's window into multiple rectangular regions. Each region contains separate html web page and each of them work independently.

A set of frames in the entire browser is known as frameset. It tells the browser how to divide browser window into frames and the web pages that each has to load.

The following table describes the various tags used for creating frames:

Tag	Description
<frameset> </frameset>	It is replacement of the <body> tag. It doesn't contain the tags that are normally used in <body> element; instead it contains the <frame> element used to add each frame.
<frame> </frame>	Specifies the content of different frames in a web page.
<base> </base>	It is used to set the default target frame in any page that contains links whose contents are displayed in another frame.

Forms

Forms are used to input the values. These values are sent to the server for processing. Forms uses input elements such as text fields, check boxes, radio buttons, lists, submit buttons etc. to enter the data into it.

The following table describes the commonly used tags while creating a form:

Tag	Description
<form> </form>	It is used to create HTML form.
<input> </input>	Specifies the input field.

<textarea> </textarea>	Specifies a text area control that allows to enter multi-line text.
<label> </label>	Specifies the label for an input element.

Marking up for structure and style

HTML stands for **H**yper**t**ext **M**arkup **L**anguage, and it is the most widely used language to write Web Pages. As its name suggests, HTML is a markup language.

- **Hypertext** refers to the way in which Web pages (HTML documents) are linked together. When you click a link in a Web page, you are using hypertext.
- **Markup Language** describes how HTML works. With a markup language, you simply "mark up" a text document with tags that tell a Web browser how to structure it to display.

Originally, HTML was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers.

All you need to do to use HTML is to learn what type of markup to use to get the results you want.

Creating HTML Document:

Creating an HTML document is easy. To begin coding HTML you need only two things: a simple-text editor and a web browser. Notepad is the most basic of simple-text editors and you will probably code a fair amount of HTML with it.

You can use our HTML Online Editor to learn HTML. Here are the simple steps to create a baisc HTML document:

- Open Notepad or another text editor.
- At the top of the page type <html>.
- On the next line, indent five spaces and now add the opening header tag: <head>.
- On the next line, indent ten spaces and type <title> </title>.
- Go to the next line, indent five spaces from the margin and insert the closing header tag: </head>.

- Five spaces in from the margin on the next line, type<body>.
- Now drop down another line and type the closing tag right below its mate: </body>.
- Finally, go to the next line and type </html>.
- In the File menu, choose Save As.
- In the Save as Type option box, choose All Files.
- Name the file template.htm.
- Click Save.

You have basic HTML document now, to see some result put the following code in title and body tags.

```
<html>
<head>
<title>This is document title</title>
</head>
<body>
<h1>This is a heading</h1>
<p>Document description goes here ....</p>
</body>
</html>
```

Now you have created one **HTML page** and you can use a Web Browser to open this HTML file to see the result. Hope you understood that Web Pages are nothing but they are simple HTML files with some content which can be rendered using Web Browsers.

Here <html>, <head>,...<p>, <h1> etc. are called HTML tags. HTML tags are building blocks of an HTML document nd we will learn all the HTML tags in subsequent chapters.

NOTE: One HTML file can have extension as **.htm** or **.html**. So you can use either of them based on your comfort.

HTML Document Structure:

An HTML document starts and ends with <html> and >/html> tags. These tags tell the browser that the entire document is composed in HTML. Inside these two tags, the document is split into two sections:

- The <head>...</head> elements, which contain information about the document such as title of the document, author of the document etc. Information inside this tag does not display outside.
- The <body>...</body> elements, which contain the real content of the document that you see on your screen.

HTML Tags and Elements:

HTML language is a markup language and we use many tags to markup text. In the above example you have seen <html>, <body> etc. are called HTML tags or HTML elements.

Every tag consists of a tag name, sometimes followed by an optional list of tag attributes , all placed between opening and closing brackets (< and >). The simplest tag is nothing more than a name appropriately enclosed in brackets, such as <head> and <i>. More complicated tags contain one or more attributes , which specify or modify the behavior of the tag.

According to the HTML standard, tag and attribute names are not case-sensitive. There's no difference in effect between <head>, <Head>, <HEAD>, or even <HeaD>; they are all equivalent. But with XHTML, case is important: all current standard tag and attribute names are in lowercase.

HTML is Forgiving?

A very good quality associated with all the browsers is that they would not give any error if you have not put any HTML tag or attribute properly. They will just ignore that tag or attribute and will apply only correct tags and attributes before displaying the result.

We can not say, HTML is forgiving because this is just a markup language and required to format documents.

Basic Page Markups

Heading Tags

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements **<h1>, <h2>, <h3>, <h4>, <h5>,** and **<h6>**. While displaying any heading, browser adds one line before and one line after that heading.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Heading Example</title>
  </head>

  <body>
    <h1>This is heading 1</h1>
    <h2>This is heading 2</h2>
    <h3>This is heading 3</h3>
    <h4>This is heading 4</h4>
```

```
    <h5>This is heading 5</h5>
    <h6>This is heading 6</h6>
  </body>

</html>
```

This will produce the following result −

Paragraph Tag

The ** ** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening and a closing tag as shown below in the example −

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Paragraph Example</title>
  </head>

  <body>
    <p>Here is a first paragraph of text.</p>
    <p>Here is a second paragraph of text.</p>
    <p>Here is a third paragraph of text.</p>
  </body>

</html>
```

This will produce the following result −

Line Break Tag

Whenever you use the **
** element, anything following it starts from the next line. This tag is an example of an **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The
 tag has a space between the characters **br** and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use
 it is not valid in XHTML.

Example

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

```
  <head>
    <title>Line Break Example</title>
  </head>

  <body>
    <p>Hello<br />
      You delivered your assignment ontime.<br />
      Thanks<br />
      Mahnaz</p>
  </body>

</html>
```

This will produce the following result −

Centering Content

You can use ** ** tag to put any content in the center of the page or any table cell.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Centring Content Example</title>
  </head>

  <body>
    <p>This text is not in the center.</p>

    <center>
      <p>This text is in the center.</p>
    </center>
  </body>

</html>
```

This will produce following result −

Horizontal Lines

Horizontal lines are used to visually break-up sections of a document. The **<hr>** tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

For example, you may want to give a line between two paragraphs as in the given example below −

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Horizontal Line Example</title>
  </head>

  <body>
    <p>This is paragraph one and should be on top</p>
    <hr />
    <p>This is paragraph two and should be at bottom</p>
  </body>

</html>
```

This will produce the following result −

Again **<hr />** tag is an example of the **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The **<hr />** element has a space between the characters **hr** and the forward slash. If you omit this space, older browsers will have trouble rendering the horizontal line, while if you miss the forward slash character and just use **<hr>** it is not valid in XHTML

Preserve Formatting

Sometimes, you want your text to follow the exact format of how it is written in the HTML document. In these cases, you can use the preformatted tag **<pre>**.

Any text between the opening **<pre>** tag and the closing **</pre>** tag will preserve the formatting of the source document.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Preserve Formatting Example</title>
  </head>

  <body>
```

```
    <pre>
      function testFunction( strText ){
        alert (strText)
      }
    </pre>
  </body>

</html>
```

This will produce the following result −

Try using the same code without keeping it inside **<pre>...</pre>** tags

Nonbreaking Spaces

Suppose you want to use the phrase "12 Angry Men." Here, you would not want a browser to split the "12, Angry" and "Men" across two lines −

An example of this technique appears in the movie "12 Angry Men."

In cases, where you do not want the client browser to break text, you should use a nonbreaking space entity ** ** instead of a normal space. For example, when coding the "12 Angry Men" in a paragraph, you should use something similar to the following code −

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Nonbreaking Spaces Example</title>
  </head>

  <body>
    <p>An example of this technique appears in the movie "12 Angry Men."</p>
  </body>

</html>
```

Absolute and Relative Link

Still today, one of the more tricky and confusing things about HTML is linking to other pages and sites, especially when absolute and relative paths come into play. But worry not! Creating links — relative and absolute alike — is actually fairly easy. Read on, and by the end of this

article, you'll know the difference between these two types of links, as well as when and how to use them.

Of course, it's still important to understand how relative and absolute links work, so read on...

First off, as you may or may not know, you would use the following code to create a link in HTML:
Click Me

linkhere.html would be the page you want to link to, and **Click Me** would be the blue, underlined link that the page displays.
In the example above, we used a relative path. You can tell if a link is relative if the path isn't a full website address. (A full website address includes **http://www.**) As you may have guessed, an absolute path does provide the full website address. Here are a few basic examples of relative and absolute paths:
Relative Paths

- index.html
- /graphics/image.png
- /help/articles/how-do-i-set-up-a-webpage.html

Absolute Paths

- http://www.mysite.com
- http://www.mysite.com/graphics/image.png
- http://www.mysite.com/help/articles/how-do-i-set-up-a-webpage.html

The first difference you'll notice between the two different types of links is that absolute paths *always* include the domain name of the website, including **http://www.**, whereas relative links only point to a file or a file path. When a user clicks a relative link, the browser takes them to that location on the current site. For that reason, you can only use relative links when linking to pages or files within your site, and you must use absolute links if you're linking to a location on another website.
So, when a user clicks a relative link, how does their browser know where to take them? Well, it looks for the location of the file *relative* to the page where the link appears. (That's where the name comes from!) Let's get back to our first example:
Click Me

This link points to a filename, with no path provided. This means that **linkhere.html** is located in the same folder as the page where this link appears. If both files were located in the root directory of the Website **http://www.website.com**, the actual website address the user would be taken to is **http://www.website.com/linkhere.html**. If both files were located in a subfolder of the root directory called **files**, the user would be taken
to **http://www.website.com/files/linkhere.html**.
How about another example? Let's say we our http://www.website.com domain had a subfolder called **pictures**. Inside the pictures folder is a file called **pictures.html**. The full path to this page would be:
"http://www.website.com/pictures/pictures.html"

Still with us? Good. Let's say in this pictures.html file, we have a link:
More Pictures

If someone clicked that, where do you think it would take them? If you said **http://www.website.com/pictures/morepictures.html**, you'd be right! You probably know why it would take them there: because both files are saved in the **pictures** subfolder.

Now, what if we wanted to use a relative link to show a page in another folder? If you want to link to a file in a subfolder of the current folder, provide the file path to that file, like so:
Read about my Tahiti vacation.

In this example, you're telling the browser to look in the current folder (**pictures**) for a subfolder (**tahiti-vacation**) that contains the file you want the user taken to (**tahiti.html**). You can link to as many subfolders as you need using this method.
What if you want to link to a file in a folder above the current folder? You have to tell the browser to move up one folder in your relative link by putting two periods and a slash (**../**) in front of the filename or path:
Learn more about my Website.

When the browser sees **../** in front of the filename, it looks in the folder above the current folder. You can use this as many times as you need to. You can also tell the browser to look in a subfolder of the directory above the current one. Using the same example website from above, let's say we wanted to create a link that would take the user to a page called **stories.html** located in another folder called **stories**. This folder is located in the root directory, one folder up from the current folder, **pictures**. Here's how a relative link to this file would look:
Read Stories

Now, let's talk about absolute paths. Like we mentioned earlier, absolute paths provide the complete website address where you want the user to go. An absolute link would look like this:
Click here to visit CoffeeCup Software.

You *must* use absolute paths when linking to another Website, but you can also use absolute paths within your own website. This practice is generally frowned upon, though. Relative links make it easy to do things like change your domain name without having to go through all your HTML pages, hunting down links and changing the names. As an added bonus, they force you to keep your site structure neat and organized, which is always a good idea.

To format both unordered and ordered lists, use the list-style-type, list-style-image, and list-style-position properties.

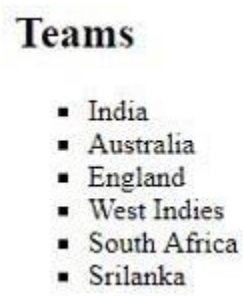
Example

Let us see an example wherein we are formatting unordered lists (ul) −

```
<!DOCTYPE html>

<html>

<head>
```

```
<style>
ul {
  list-style-type: square;
}
</style>
</head>
<body>
<h2>Teams</h2>
<ul>
<li>India</li>
<li>Australia</li>
<li>England</li>
<li>West Indies</li>
<li>South Africa</li>
<li>Srilanka</li>
</ul>
</body>
</html>
```

Output



Example

Let us now see an example wherein we are formatting ordered lists (ol) −

```
<!DOCTYPE html>
<html>
<head>
<style>
ol {
   list-style-type: upper-roman;
}
</style>
</head>
<body>
<h2>Teams</h2>
<ol>
<li>India</li>
<li>Australia</li>
<li>England</li>
<li>West Indies</li>
<li>South Africa</li>
<li>Srilanka</li>
</ol>
</body>
</html>
```

Output

Example

Let us see another example wherein we will set an image for list style for both ordered and unordered lists −

```
<!DOCTYPE html>
<html>
<head>
<style>
ul.demo1 {
   list-style-image: url('https://www.tutorialspoint.com/images/Swift.png');
}
ol.demo2 {
   list-style-image: url('https://www.tutorialspoint.com/images/Swift.png');
}
</style>
</head>
<body>
<h2>Teams</h2>
<ul class="demo1">
```

```
<li>India - Qualified for WordCup</li>
<li>Australia - Qualified for WordCup</li>
<li>England - Qualified for WordCup</li>
</ul>
<h2>Players</h2>
<ol class="demo2">
<li>Virat Kohli</li>
<li>David Warner</li>
<li>Steve Smith</li>
</ol>
</body>
</html>
```

Output

Embedding images and controlling Appearance

Insert Image

You can insert any image in your web page by using **<img>** tag. Following is the simple syntax to use this tag.

The <img> tag is an empty tag, which means that, it can contain only list of attributes and it has no closing tag.

Example

To try following example, let's keep our HTML file test.htm and image file test.png in the same directory −

```
<!DOCTYPE html>
<html>

  <head>
    <title>Using Image in Webpage</title>
  </head>

  <body>
    <p>Simple Image Insert</p>
    <img src = "/html/images/test.png" alt = "Test Image" />
  </body>

</html>
```

This will produce the following result −

You can use PNG, JPEG or GIF image file based on your comfort but make sure you specify correct image file name in **src** attribute. Image name is always case sensitive.

The **alt** attribute is a mandatory attribute which specifies an alternate text for an image, if the image cannot be displayed.

Set Image Location

Usually we keep all the images in a separate directory. So let's keep HTML file test.htm in our home directory and create a subdirectory **images** inside the home directory where we will keep our image test.png.

Example

Assuming our image location is "image/test.png", try the following example −

```
<!DOCTYPE html>
<html>

  <head>
    <title>Using Image in Webpage</title>
  </head>

  <body>
    <p>Simple Image Insert</p>
    <img src = "/html/images/test.png" alt = "Test Image" />
  </body>

</html>
```

This will produce the following result −

Set Image Width/Height

You can set image width and height based on your requirement using **width** and **height** attributes. You can specify width and height of the image in terms of either pixels or percentage of its actual size.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Set Image Width and Height</title>
  </head>

  <body>
    <p>Setting image width and height</p>
    <img src = "/html/images/test.png" alt = "Test Image" width = "150" height = "100"/>
  </body>

</html>
```

This will produce the following result −

Set Image Border

By default, image will have a border around it, you can specify border thickness in terms of pixels using border attribute. A thickness of 0 means, no border around the picture.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Set Image Border</title>
  </head>

  <body>
    <p>Setting image Border</p>
    <img src = "/html/images/test.png" alt = "Test Image" border = "3"/>
  </body>
```

```
</html>
```

This will produce the following result −

Set Image Alignment

By default, image will align at the left side of the page, but you can use **align** attribute to set it in the center or right.

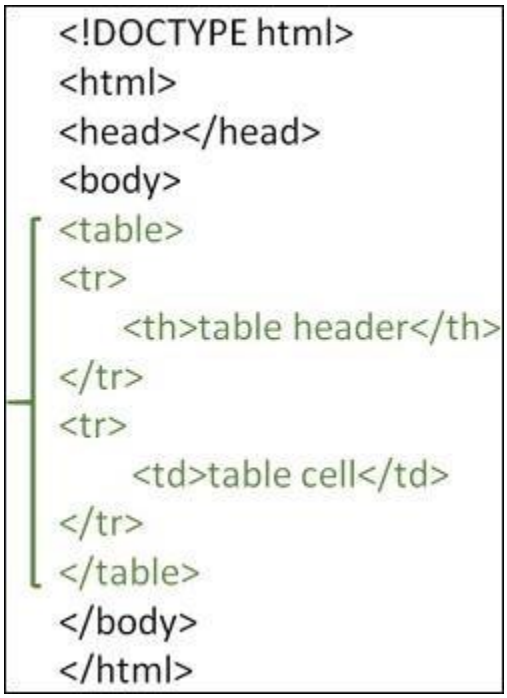
Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Set Image Alignment</title>
  </head>

  <body>
    <p>Setting image Alignment</p>
    <img src = "/html/images/test.png" alt = "Test Image" border = "3" align = "right"/>
  </body>

</html>
```

Table creation and use

To create table in HTML, use the <table> tag. A table consist of rows and columns, which can be set using one or more <tr>, <th>, and <td> elements. A table row is defined by the <tr> tag. To set table header, use the <th> tag. For a table cell, use the <td> tag.

Just keep in mind, table attributes such as align, bgcolor, border, cellpadding, cellspacing deprecated and isn't supported by HTML5. Do not use them.



You can try the following code to create a table in HTML. We're also using the <style> tag to style the table border

Example

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

  <body>
    <h1>Programming Languages</h1>
    <table>
      <tr>
```

```
        <th>Language</th>
        <th>Release Year</th>
      </tr>
      <tr>
        <td>Java</td>
        <td>1995</td>
      </tr>
      <tr>
        <td>Pascal</td>
        <td>1970</td>
      </tr>
    </table>
  </body>
</html>
```

HTML Frames

HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Disadvantages of Frames

There are few drawbacks with using frames, so it's never recommended to use frames in your webpages −

- Some smaller devices cannot cope with frames often because their screen is not big enough to be divided up.
- Sometimes your page will be displayed differently on different computers due to different screen resolution.

- The browser's back button might not work as the user hopes.
- There are still few browsers that do not support frame technology.

Creating Frames

To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines, how to divide the window into frames. The **rows** attribute of <frameset> tag defines horizontal frames and **cols** attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Note − The <frame> tag deprecated in HTML5. Do not use this element.

Example

Following is the example to create three horizontal frames −

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Frames</title>
  </head>

  <frameset rows = "10%,80%,10%">
    <frame name = "top" src = "/html/top_frame.htm" />
    <frame name = "main" src = "/html/main_frame.htm" />
    <frame name = "bottom" src = "/html/bottom_frame.htm" />

    <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>

  </frameset>

</html>
```

This will produce the following result −

Example

Let's put the above example as follows, here we replaced rows attribute by cols and changed their width. This will create all the three frames vertically −

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

```
  <head>
    <title>HTML Frames</title>
  </head>

  <frameset cols = "25%,50%,25%">
    <frame name = "left" src = "/html/top_frame.htm" />
    <frame name = "center" src = "/html/main_frame.htm" />
    <frame name = "right" src = "/html/bottom_frame.htm" />

    <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>
  </frameset>

</html>
```

This will produce the following result −

The <frameset> Tag Attributes

Following are important attributes of the <frameset> tag −

Sr.No	Attribute & Description
1	**cols** Specifies how many columns are contained in the frameset and the size of each column. You can specify the width of each column in one of the four ways − Absolute values in pixels. For example, to create three vertical frames, use cols = "100, 500, 100". A percentage of the browser window. For example, to create three vertical frames, use cols = "10%, 80%, 10%". Using a wildcard symbol. For example, to create three vertical frames, use cols = "10%, *, 10%". In this case wildcard takes remainder of the window. As relative widths of the browser window. For example, to create three vertical frames, use cols = "3*, 2*, 1*". This is an alternative to percentages. You can use relative widths of the browser window. Here the window is divided into sixths: the first column takes up half of the window, the second takes one third, and the third takes one sixth.

2	**rows** This attribute works just like the cols attribute and takes the same values, but it is used to specify the rows in the frameset. For example, to create two horizontal frames, use rows = "10%, 90%". You can specify the height of each row in the same way as explained above for columns.
3	**border** This attribute specifies the width of the border of each frame in pixels. For example, border = "5". A value of zero means no border.
4	**frameborder** This attribute specifies whether a three-dimensional border should be displayed between frames. This attribute takes value either 1 (yes) or 0 (no). For example frameborder = "0" specifies no border.
5	**framespacing** This attribute specifies the amount of space between frames in a frameset. This can take any integer value. For example framespacing = "10" means there should be 10 pixels spacing between each frames.

The <frame> Tag Attributes

Following are the important attributes of <frame> tag −

Sr.No	Attribute & Description
1	**src** This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src = "/html/top_frame.htm" will load an HTML file available in html directory.
2	**name** This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame, in which case the second frame needs a name to identify itself as the target of the link.

3	**frameborder** This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
4	**marginwidth** This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example marginwidth = "10".
5	**marginheight** This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example marginheight = "10".
6	**noresize** By default, you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example noresize = "noresize".
7	**scrolling** This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example scrolling = "no" means it should not have scroll bars.
8	**longdesc** This attribute allows you to provide a link to another page containing a long description of the contents of the frame. For example longdesc = "framedescription.htm"

Browser Support for Frames

If a user is using any old browser or any browser, which does not support frames then <noframes> element should be displayed to the user.

So you must place a <body> element inside the <noframes> element because the <frameset> element is supposed to replace the <body> element, but if a browser does not understand

<frameset> element then it should understand what is inside the <body> element which is contained in a <noframes> element.

You can put some nice message for your user having old browsers. For example, Sorry!! your browser does not support frames. as shown in the above example.

Frame's name and target attributes

One of the most popular uses of frames is to place navigation bars in one frame and then load main pages into a separate frame.

Let's see following example where a test.htm file has following code −

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Target Frames</title>
  </head>

  <frameset cols = "200, *">
    <frame src = "/html/menu.htm" name = "menu_page" />
    <frame src = "/html/main.htm" name = "main_page" />

    <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>
  </frameset>

</html>
```

Here, we have created two columns to fill with two frames. The first frame is 200 pixels wide and will contain the navigation menu bar implemented by **menu.htm** file. The second column fills in remaining space and will contain the main part of the page and it is implemented by **main.htm** file. For all the three links available in menu bar, we have mentioned target frame as **main_page**, so whenever you click any of the links in menu bar, available link will open in main page.

Following is the content of menu.htm file

```
<!DOCTYPE html>
<html>

  <body bgcolor = "#4a7d49">
    <a href = "http://www.google.com" target = "main_page">Google</a>
```

```
    <br />
    <br />

    <a href = "http://www.microsoft.com" target = "main_page">Microsoft</a>
    <br />
    <br />

    <a href = "http://news.bbc.co.uk" target = "main_page">BBC News</a>
  </body>

</html>
```

Following is the content of main.htm file −

```
<!DOCTYPE html>
<html>

  <body bgcolor = "#b5dcb3">
    <h3>This is main page and content from any link will be displayed here.</h3>
    <p>So now click any link and see the result.</p>
  </body>

</html>
```

When we load **test.htm** file, it produces following result −

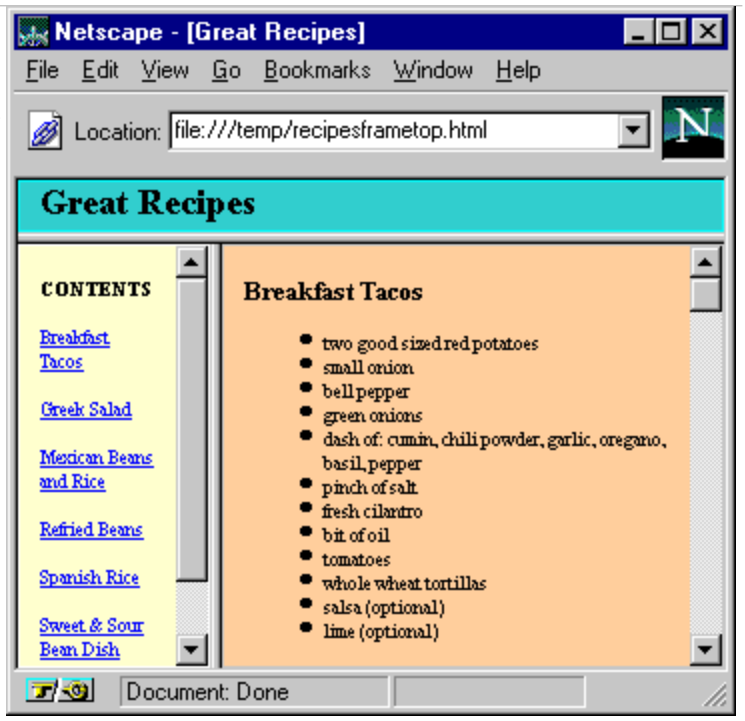
Now you can try to click links available in the left panel and see the result. The targetattribute can also take one of the following values −

Sr.No	Option & Description
1	**_self** Loads the page into the current frame.
2	**_blank** Loads a page into a new browser window. Opening a new window.
3	**_parent** Loads the page into the parent window, which in the case of a single frameset is the main browser window.

4	**_top** Loads the page into the browser window, replacing any current frames.
5	**targetframe** Loads the page into a named targetframe.

Nesting

Nested Framesets



Let's move now to a more real world example, and a few more techniques for using frames. One of the most popular uses for frames is the "title bar and side menu" method. We'll use as an example a page of recipes, pictured at right. The title of the page, "Great Recipes" stays stationary in a frame at top, a contents list is on the left, and the recipes themselves are in the large box on the right. As you click on the name of a recipe in the contents list, that recipe appears on the right. Go ahead and try out the real page. (We're sorry if these recipes make you hungry. They did us. These recipes come from the wonderful vegetarian recipe site Veggies Unite!.)

Remember that a frameset is like a "table of documents" with rows and columns. The recipes page, however, has one column on top, but two on bottom. This is done by nesting framesets, putting one frameset inside another.

Here's the code for the frameset file for the recipes page:

```
<HTML>
<HEAD>
<TITLE>Great Recipes</TITLE>
</HEAD>
```

```
<FRAMESET ROWS="15%,*">
  <FRAME SRC="recipetitlebar.html" NAME=TITLE SCROLLING=NO>

  <FRAMESET COLS="20%,*">
    <FRAME SRC="recipesidebar.html" NAME=SIDEBAR>
    <FRAME SRC="recipes.html" NAME=RECIPES>
  </FRAMESET>

  <NOFRAMES>
  <H1>Great Recipes</H1>
  No frames? No Problem! Take a look at our
  <A HREF="recipes.html">no-frames</A> version.
  </NOFRAMES>

</FRAMESET>

</HTML>
```

The first <FRAMESET ...> tag says "this frameset will have two rows" (and, implicitly, only one column, since COLS was left out). The first <FRAME ...> tag puts a document in the first frame. The second frame is filled in not by a document but by another frameset. The second <FRAMESET ...> is creating a "table within a table", or, to be more correct, a frameset within a frameset.

Targetting

HTML <a> target Attribute

❮ HTML <a> tag

Example

The target attribute specifies where to open the linked document:

```
<a href="https://www.w3schools.com" target="_blank">Visit W3Schools</a>
```

Definition and Usage

The target attribute specifies where to open the linked document.

Browser Support

Attribute					
Target	Yes	Yes	Yes	Yes	Yes

Syntax

<a target="_blank|_self|_parent|_top|framename">

Attribute Values

Value	**Description**
_blank	Opens the linked document in a new window or tab
_self	Opens the linked document in the same frame as it was clicked (this is default)
_parent	Opens the linked document in the parent frame
_top	Opens the linked document in the full body of the window
Framename	Opens the linked document in a named frame

<!DOCTYPE html>

<html>

<body>

<h1>The a target attribute</h1>

** Open link in a new window or tab: Visit W3Schools! **

</body>

</html>

The a target attribute

Open link in a new window or tab: Visit W3Schools!

HTML meta tags

HTML lets you specify metadata - additional important information about a document in a variety of ways. The META elements can be used to include name/value pairs describing properties of the HTML document, such as author, expiry date, a list of keywords, document author etc.

The **<meta>** tag is used to provide such additional information. This tag is an empty element and so does not have a closing tag but it carries information within its attributes.

You can include one or more meta tags in your document based on what information you want to keep in your document but in general, meta tags do not impact physical appearance of the document so from appearance point of view, it does not matter if you include them or not.

Adding Meta Tags to Your Documents

You can add metadata to your web pages by placing <meta> tags inside the header of the document which is represented by **<head>** and **</head>** tags. A meta tag can have following attributes in addition to core attributes −

Sr.No	Attribute & Description
1	**Name** Name for the property. Can be anything. Examples include, keywords,

	description, author, revised, generator etc.
2	**Content** Specifies the property's value.
3	**Scheme** Specifies a scheme to interpret the property's value (as declared in the content attribute).
4	**http-equiv** Used for http response message headers. For example, http-equiv can be used to refresh the page or to set a cookie. Values include content-type, expires, refresh and set-cookie.

Specifying Keywords

You can use <meta> tag to specify important keywords related to the document and later these keywords are used by the search engines while indexing your webpage for searching purpose.

Example

Following is an example, where we are adding HTML, Meta Tags, Metadata as important keywords about the document.

```
<!DOCTYPE html>
<html>

  <head>
    <title>Meta Tags Example</title>
    <meta name = "keywords" content = "HTML, Meta Tags, Metadata" />
  </head>

  <body>
    <p>Hello HTML5!</p>
  </body>

</html>
```

This will produce the following result −

Document Description

You can use <meta> tag to give a short description about the document. This again can be used by various search engines while indexing your webpage for searching purpose.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Meta Tags Example</title>
    <meta name = "keywords" content = "HTML, Meta Tags, Metadata" />
    <meta name = "description" content = "Learning about Meta Tags." />
  </head>

  <body>
    <p>Hello HTML5!</p>
  </body>

</html>
```

Document Revision Date

You can use <meta> tag to give information about when last time the document was updated. This information can be used by various web browsers while refreshing your webpage.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Meta Tags Example</title>
    <meta name = "keywords" content = "HTML, Meta Tags, Metadata" />
    <meta name = "description" content = "Learning about Meta Tags." />
    <meta name = "revised" content = "Tutorialspoint, 3/7/2014" />
  </head>

  <body>
    <p>Hello HTML5!</p>
  </body>

</html>
```

Document Refreshing

A <meta> tag can be used to specify a duration after which your web page will keep refreshing automatically.

Example

If you want your page keep refreshing after every 5 seconds then use the following syntax.

```
<!DOCTYPE html>
<html>

  <head>
    <title>Meta Tags Example</title>
    <meta name = "keywords" content = "HTML, Meta Tags, Metadata" />
    <meta name = "description" content = "Learning about Meta Tags." />
    <meta name = "revised" content = "Tutorialspoint, 3/7/2014" />
    <meta http-equiv = "refresh" content = "5" />
  </head>

  <body>
    <p>Hello HTML5!</p>
  </body>

</html>
```

Page Redirection

You can use <meta> tag to redirect your page to any other webpage. You can also specify a duration if you want to redirect the page after a certain number of seconds.

Example

Following is an example of redirecting current page to another page after 5 seconds. If you want to redirect page immediately then do not specify content attribute.

```
<!DOCTYPE html>
<html>

  <head>
    <title>Meta Tags Example</title>
    <meta name = "keywords" content = "HTML, Meta Tags, Metadata" />
    <meta name = "description" content = "Learning about Meta Tags." />
    <meta name = "revised" content = "Tutorialspoint, 3/7/2014" />
    <meta http-equiv = "refresh" content = "5; url = http://www.tutorialspoint.com" />
  </head>

  <body>
    <p>Hello HTML5!</p>
  </body>
```

```
</html>
```

Setting Cookies

Cookies are data, stored in small text files on your computer and it is exchanged between web browser and web server to keep track of various information based on your web application need.

You can use <meta> tag to store cookies on client side and later this information can be used by the Web Server to track a site visitor.

Example

Following is an example of redirecting current page to another page after 5 seconds. If you want to redirect page immediately then do not specify content attribute.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Meta Tags Example</title>
    <meta http-equiv = "cookie" content = "userid = xyz; expires = Wednesday, 08-Aug-15
23:59:59 GMT;" />

  </head>
  <body>
    <p>Hello HTML5!</p>
  </body>
</html>
```

If you do not include the expiration date and time, the cookie is considered a session cookie and will be deleted when the user exits the browser.

Note − You can check PHP and Cookies tutorial for a complete detail on Cookies.

Setting Author Name

You can set an author name in a web page using meta tag. See an example below −

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Meta Tags Example</title>
    <meta name = "keywords" content = "HTML, Meta Tags, Metadata" />
    <meta name = "description" content = "Learning about Meta Tags." />
    <meta name = "author" content = "Mahnaz Mohtashim" />
  </head>
```

```
  <body>
    <p>Hello HTML5!</p>
  </body>

</html>
```

Specify Character Set

You can use <meta> tag to specify character set used within the webpage.

Example

By default, Web servers and Web browsers use ISO-8859-1 (Latin1) encoding to process Web pages. Following is an example to set UTF-8 encoding −

```
<!DOCTYPE html>
<html>

  <head>
    <title>Meta Tags Example</title>
    <meta name = "keywords" content = "HTML, Meta Tags, Metadata" />
    <meta name = "description" content = "Learning about Meta Tags." />
    <meta name = "author" content = "Mahnaz Mohtashim" />
    <meta http-equiv = "Content-Type" content = "text/html; charset = UTF-8" />
  </head>

  <body>
    <p>Hello HTML5!</p>
  </body>

</html>
```

To serve the static page with traditional Chinese characters, the webpage must contain a <meta> tag to set Big5 encoding −

```
<!DOCTYPE html>
<html>

  <head>
    <title>Meta Tags Example</title>
    <meta name = "keywords" content = "HTML, Meta Tags, Metadata" />
    <meta name = "description" content = "Learning about Meta Tags." />
    <meta name = "author" content = "Mahnaz Mohtashim" />
    <meta http-equiv = "Content-Type" content = "text/html; charset = Big5" />
```

```
  </head>

  <body>
    <p>Hello HTML5!</p>
  </body>

</html>
```

HTML Semantic Elements

Semantic elements = elements with a meaning.

What are Semantic Elements?

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of **non-semantic** elements: `<div>` and `<span>` - Tells nothing about its content.

Examples of **semantic** elements: `<form>`, `<table>`, and `<article>` - Clearly defines its content.

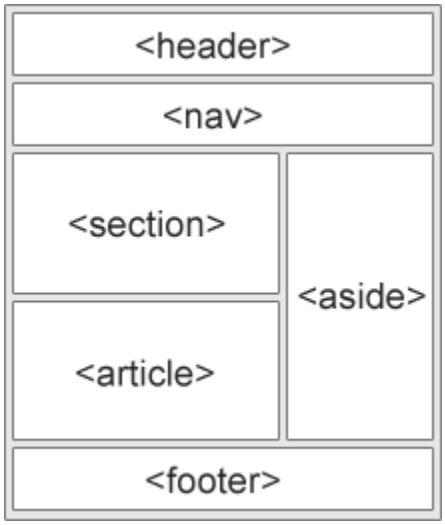
Semantic Elements in HTML

Many web sites contain HTML code like: to indicate navigation, header, and footer.

In HTML there are some semantic elements that can be used to define different parts of a web page:

- <article>
- <aside>
- <details>
- <figcaption>
- <figure>
- <footer>

- <header>
- <main>
- <mark>
- <nav>
- <section>
- <summary>
- <time>



HTML <section> Element

The <section> element defines a section in a document.

According to W3C's HTML documentation: "A section is a thematic grouping of content, typically with a heading."

A home page could normally be split into sections for introduction, content, and contact information.

Example

```
<section>
  <h1>WWF</h1>
  <p>The World Wide Fund for Nature (WWF) is ... </p>
</section>
```

HTML <article> Element

The <article> element specifies independent, self-contained content.

An article should make sense on its own, and it should be possible to read it independently from the rest of the web site.

Examples of where an <article> element can be used:

- Forum post
- Blog post
- Newspaper article

Example

```
<article>
  <h1>What Does WWF Do?</h1>
  <p>WWF's mission is to stop the degradation of our planet's natural environment,
  and build a future in which humans live in harmony with nature.</p>
</article>
```

Nesting <article> in <section> or Vice Versa?

The<article> element specifies independent, self-contained content.

The <section> element defines section in a document.

Can we use the definitions to decide how to nest those elements? No, we cannot!

So, on the Internet, you will find HTML pages with <section> elements containing <article> elements, and <article> elements containing <section> elements.

You will also find pages with <section> elements containing <section> elements, and <article> elements containing <article> elements.

Example for a newspaper: The sport <article> in the sport **section**, may have a technical **section** in each <article>.

HTML <header> Element

The <header> element specifies a header for a document or section.

The <header> element should be used as a container for introductory content.

You can have several <header> elements in one document.

The following example defines a header for an article:

Example

```
<article>
  <header>
    <h1>What Does WWF Do?</h1>
    <p>WWF's mission:</p>
  </header>
  <p>WWF's mission is to stop the degradation of our planet's natural environment,
  and build a future in which humans live in harmony with nature.</p>
</article>
```

HTML <footer> Element

The <footer> element specifies a footer for a document or section.

A <footer> element should contain information about its containing element.

A footer typically contains the author of the document, copyright information, links to terms of use, contact information, etc.

You may have several <footer> elements in one document.

Example

```
<footer>
  <p>Posted by: Hege Refsnes</p>
  <p>Contact information: <a href="mailto:someone@example.com">
  someone@example.com</a>.</p>
</footer>
```

HTML <nav> Element

The <nav> element defines a set of navigation links.

Notice that NOT all links of a document should be inside a <nav> element. The <nav> element is intended only for major block of navigation links.

Example

```
<nav>
  <a href="/html/">HTML</a> |
  <a href="/css/">CSS</a> |
  <a href="/js/">JavaScript</a> |
  <a href="/jquery/">jQuery</a>
</nav>
```

HTML <aside> Element

The <aside> element defines some content aside from the content it is placed in (like a sidebar).

The <aside> content should be related to the surrounding content.

Example

```
<p>My family and I visited The Epcot center this summer.</p>

<aside>
  <h4>Epcot Center</h4>
  <p>The Epcot Center is a theme park in Disney World, Florida.</p>
</aside>
```

HTML <figure> and <figcaption> Elements

An image and a caption can be grouped together in a <figure> element.

The purpose of a caption is to add a visual explanation to an image.

Example

```
<figure>
  <img src="pic_trulli.jpg" alt="Trulli">
  <figcaption>Fig1. - Trulli, Puglia, Italy.</figcaption>
</figure>
```

The <img> element defines the image, the <figcaption> element defines the caption.

Why Semantic Elements?

According to the W3C: "A semantic Web allows data to be shared and reused across applications, enterprises, and communities."

Semantic Elements in HTML

Below is an alphabetical list of some of the semantic elements in HTML.

The links go to our complete HTML Reference.

Tag	Description
<article>	Defines an article
<aside>	Defines content aside from the page content
<details>	Defines additional details that the user can view or hide
<figcaption>	Defines a caption for a <figure> element
<figure>	Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
<footer>	Defines a footer for a document or section
<header>	Specifies a header for a document or section

<main>	Specifies the main content of a document
<mark>	Defines marked/highlighted text
<nav>	Defines navigation links
<section>	Defines a section in a document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time

Doubling code & RDF

XML RDF

❮ PreviousNext ❯

RDF Document Example

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

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:si="https://www.w3schools.com/rdf/">

<rdf:Description rdf:about="https://www.w3schools.com">
```

```
  <si:title>W3Schools</si:title>
  <si:author>Jan Egil Refsnes</si:author>
</rdf:Description>

</rdf:RDF>
```

What is RDF?

- RDF stands for **R**esource **D**escription **F**ramework
- RDF is a framework for describing resources on the web
- RDF is designed to be read and understood by computers
- RDF is not designed for being displayed to people
- RDF is written in XML
- RDF is a part of the W3C's Semantic Web Activity
- RDF is a W3C Recommendation from 10. February 2004

RDF - Examples of Use

- Describing properties for shopping items, such as price and availability
- Describing time schedules for web events
- Describing information about web pages (content, author, created and modified date)
- Describing content and rating for web pictures
- Describing content for search engines
- Describing electronic libraries

RDF is Designed to be Read by Computers

RDF was designed to provide a common way to describe information so it can be read and understood by computer applications.

RDF descriptions are not designed to be displayed on the web.

RDF is Written in XML

RDF documents are written in XML. The XML language used by RDF is called RDF/XML.

By using XML, RDF information can easily be exchanged between different types of computers using different types of operating systems and application languages.

RDF and "The Semantic Web"

The RDF language is a part of the W3C's Semantic Web Activity. W3C's "Semantic Web Vision" is a future where:

- Web information has exact meaning
- Web information can be understood and processed by computers
- Computers can integrate information from the web

RDF uses Web identifiers (URIs) to identify resources.

RDF describes resources with properties and property values.

RDF Resource, Property, and Property Value

RDF identifies things using Web identifiers (URIs), and describes resources with properties and property values.

Explanation of Resource, Property, and Property value:

- A **Resource** is anything that can have a URI, such as "https://www.w3schools.com/rdf"
- A **Property** is a Resource that has a name, such as "author" or "homepage"
- A **Property value** is the value of a Property, such as "Jan Egil Refsnes" or "https://www.w3schools.com" (note that a property value can be another resource)

The following RDF document could describe the resource "https://www.w3schools.com/rdf":

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

<RDF>
  <Description about="https://www.w3schools.com/rdf">
    <author>Jan Egil Refsnes</author>
    <homepage>https://www.w3schools.com</homepage>
  </Description>
</RDF>
```

The example above is simplified. Namespaces are omitted.

RDF Statements

The combination of a Resource, a Property, and a Property value forms a **Statement** (known as the **subject, predicate and object** of a Statement).

Let's look at some example statements to get a better understanding:

Statement: "The author of https://www.w3schools.com/rdf is Jan Egil Refsnes".

- The subject of the statement above is: https://www.w3schools.com/rdf
- The predicate is: author
- The object is: Jan Egil Refsnes

Statement: "The homepage of https://www.w3schools.com/rdf is https://www.w3schools.com".

- The subject of the statement above is: https://www.w3schools.com/rdf
- The predicate is: homepage
- The object is: https://www.w3schools.com

UNIT-2

Separating style from structure with style sheets:

Internal style specification within HTML

External linked style specification using CSS

Cascading Style Sheets (CSS) describe how documents are presented on screens, in print, or perhaps how they are pronounced. W3C has actively promoted the use of style sheets on the Web since the consortium was founded in 1994.

Cascading Style Sheets (CSS) provide easy and effective alternatives to specify various attributes for the HTML tags. Using CSS, you can specify a number of style properties for a given HTML element. Each property has a name and a value, separated by a colon (:). Each property declaration is separated by a semi-colon (;).

Example

First let's consider an example of HTML document which makes use of <font> tag and associated attributes to specify text color and font size −

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML CSS</title>
  </head>

  <body>
    <p><font color = "green" size = "5">Hello, World!</font></p>
  </body>

</html>
```

We can re-write above example with the help of Style Sheet as follows −

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML CSS</title>
  </head>

  <body>
    <p style = "color:green; font-size:24px;" >Hello, World!</p>
  </body>

</html>
```

This will produce the following result −

You can use CSS in three ways in your HTML document −

- **External Style Sheet** − Define style sheet rules in a separate .css file and then include that file in your HTML document using HTML <link> tag.
- **Internal Style Sheet** − Define style sheet rules in header section of the HTML document using <style> tag.
- **Inline Style Sheet** − Define style sheet rules directly along-with the HTML elements using **style** attribute.

Let's see all the three cases one by one with the help of suitable examples.

External Style Sheet

If you need to use your style sheet to various pages, then its always recommended to define a common style sheet in a separate file. A cascading style sheet file will have extension as **.css** and it will be included in HTML files using <link> tag.

Example

Consider we define a style sheet file **style.css** which has following rules −

```
.red {
  color: red;
}
.thick {
  font-size:20px;
}
.green {
```

```
  color:green;
}
```

Here we defined three CSS rules which will be applicable to three different classes defined for the HTML tags. I suggest you should not bother about how these rules are being defined because you will learn them while studying CSS. Now let's make use of the above external CSS file in our following HTML document −

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML External CSS</title>
    <link rel = "stylesheet" type = "text/css" href = "/html/style.css">
  </head>

  <body>
    <p class = "red">This is red</p>
    <p class = "thick">This is thick</p>
    <p class = "green">This is green</p>
    <p class = "thick green">This is thick and green</p>
  </body>

</html>
```

This will produce the following result −

Internal Style Sheet

If you want to apply Style Sheet rules to a single document only, then you can include those rules in header section of the HTML document using <style> tag.

Rules defined in internal style sheet overrides the rules defined in an external CSS file.

Example

Let's re-write above example once again, but here we will write style sheet rules in the same HTML document using <style> tag −

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Internal CSS</title>

    <style type = "text/css">
      .red {
        color: red;
      }
      .thick{
        font-size:20px;
      }
      .green {
        color:green;
      }
    </style>
  </head>

  <body>
    <p class = "red">This is red</p>
    <p class = "thick">This is thick</p>
    <p class = "green">This is green</p>
    <p class = "thick green">This is thick and green</p>
  </body>

</html>
```

This will produce the following result −

Inline Style Sheet

You can apply style sheet rules directly to any HTML element using **style** attribute of the relevant tag. This should be done only when you are interested to make a particular change in any HTML element only.

Rules defined inline with the element overrides the rules defined in an external CSS file as well as the rules defined in <style> element.

Example

Let's re-write above example once again, but here we will write style sheet rules along with the HTML elements using **style** attribute of those elements.

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Inline CSS</title>
  </head>

  <body>
    <p style = "color:red;">This is red</p>
    <p style = "font-size:20px;">This is thick</p>
    <p style = "color:green;">This is green</p>
    <p style = "color:green;font-size:20px;">This is thick and green</p>
  </body>

</html>
```

Page and site design considerations

Understand the medium

Readers experience Web pages in two ways: as a direct medium where pages are read online and as a delivery medium to access information that is downloaded into text files or printed onto paper. Your expectations about how readers will typically use your site should govern your page design decisions. Documents to be read online should be concise, with the amount of graphics carefully "tuned" to the bandwidth available to your mainstream audience. Documents that will most likely be printed and read offline should appear on one page, and the page width should be narrow enough to print easily on standard paper sizes.

Include fixed page elements

Each page should contain a title, an author, an institutional affiliation, a revision date, copyright information, and a link to the "home page" of your site. Web pages are often printed or saved to disk, and without this information there is no easy way to determine where the document originated. Think of each page in your site as a newspaper clipping, and make sure that the information required to determine its provenance is included.

Don't impose style

Don't set out to develop a "style" for your site, and be careful about simply importing the graphic elements of another Web site or print publication to "decorate" your pages. The graphic and editorial style of your Web site should evolve as a natural consequence of consistent and appropriate handling of your content and page layout.

Maximize prime real estate

In page layout the top of the page is always the most dominant location, but on Web pages the upper page is especially important, because the top four inches of the page are all that is visible on the typical display screen. Use this space efficiently and effectively.

Use subtle colors

Subtle pastel shades of colors typically found in nature make the best choices for background or minor elements. Avoid bold, highly saturated primary colors except in regions of maximum emphasis, and even there use them cautiously.

Beware of graphic embellishments

Horizontal rules, graphic bullets, icons, and other visual markers have their occasional uses, but apply each sparingly (if at all) to avoid a patchy and confusing layout. The same consideration applies to the larger sizes of type on Web pages. One reason professional graphic designers are so impatient with plain HTML is that the `H1` and `H2` header tags display in grotesquely large type on most Web browsers. The tools of graphic emphasis are powerful and should be used only in small doses for maximum effect. Overuse of graphic emphasis leads to a "clown's pants" effect in which everything is garish and nothing is emphasized.

JAVA SCRIPT SYNTAX

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

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 alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script           ...>
  JavaScript code
</script>
```

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>
```

Your First JavaScript Code

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

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

```
</html>
```

This code will produce the following result −

Hello World!

Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language = "javascript" type = "text/javascript">
  <!--
     var1 = 10
     var2 = 20
  //-->
</script>
```

But when formatted in a single line as follows, you must use semicolons −

```
<script language = "javascript" type = "text/javascript">
  <!--
     var1 = 10; var2 = 20;
  //-->
</script>
```

Note − It is a good programming practice to use semicolons.

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.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

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

The following example shows how to use comments in JavaScript.

```
<script language = "javascript" type = "text/javascript">
  <!--
    // This is a comment. It is similar to comments in C++

    /*
    * This is a multi-line comment in JavaScript
    * It is very similar to comments in C Programming
    */
  //-->
</script>
```

JavaScript - Document Object Model or DOM

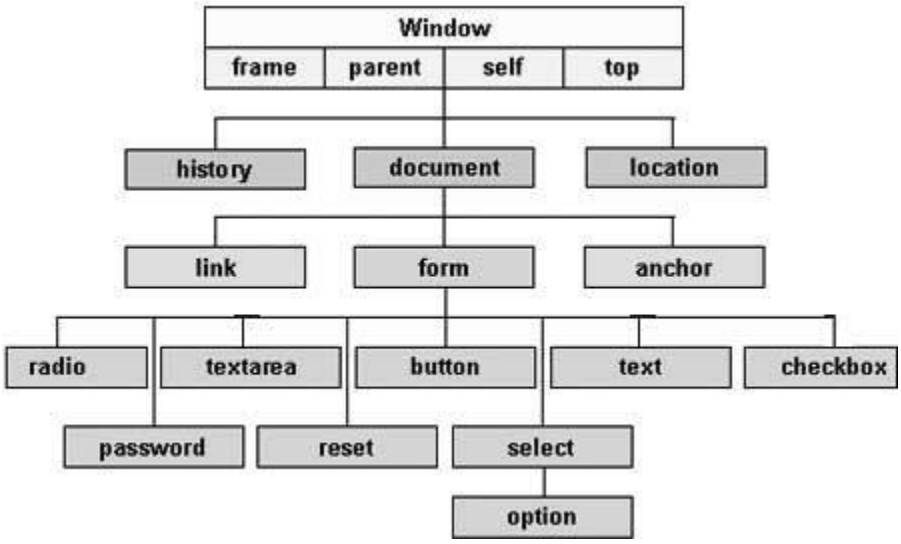
Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects −



There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

- The Legacy DOM − This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but

allows access only to certain key portions of documents, such as forms, form elements, and images.

- The W3C DOM − This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.
- The IE4 DOM − This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

DOM compatibility

If you want to write a script with the flexibility to use either W3C DOM or IE 4 DOM depending on their availability, then you can use a capability-testing approach that first checks for the existence of a method or property to determine whether the browser has the capability you desire. For example −

```
if (document.getElementById) {
   // If the W3C method exists, use it
} else if (document.all) {
   // If the all[] array exists, use it
} else {
   // Otherwise use the legacy DOM
}
```

Event Handling

What is an Event?

Change in the state of an object is known as **Event**, i.e., event describes the change in the state of the source. Events are generated as a result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from the list, and scrolling the page are the activities that causes an event to occur.

Types of Event

The events can be broadly classified into two categories −

- **Foreground Events** − These events require direct interaction of the user. They are generated as consequences of a person interacting with the graphical components in the Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.
- **Background Events** − These events require the interaction of the end user. Operating system interrupts, hardware or software failure, timer expiration, and operation completion are some examples of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler, that is executed when an event occurs.

Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The Delegation Event Model has the following key participants.

- **Source** − The source is an object on which the event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide us with classes for the source object.
- **Listener** − It is also known as event handler. The listener is responsible for generating a response to an event. From the point of view of Java implementation, the listener is also an object. The listener waits till it receives an event. Once the event is received, the listener processes the event and then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to a separate piece of code.

In this model, the listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listeners who want to receive them.

Steps Involved in Event Handling

Step 1 − The user clicks the button and the event is generated.

Step 2 − The object of concerned event class is created automatically and information about the source and the event get populated within the same object.

Step 3 − Event object is forwarded to the method of the registered listener class.

Step 4 − The method is gets executed and returns.

Points to Remember About the Listener

- In order to design a listener class, you have to develop some listener interfaces. These Listener interfaces forecast some public abstract callback methods, which must be implemented by the listener class.
- If you do not implement any of the predefined interfaces, then your class cannot act as a listener class for a source object.

Callback Methods

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represent an event method. In response to an event, java jre will fire callback method. All such callback methods are provided in listener interfaces.

If a component wants some listener to listen ot its events, the source must register itself to the listener.

Event Handling Example

Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SwingControlDemo {
   private JFrame mainFrame;
```

```
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
     prepareGUI();
   }
   public static void main(String[] args){
     SwingControlDemo swingControlDemo = new SwingControlDemo();
     swingControlDemo.showEventDemo();
   }
   private void prepareGUI(){
     mainFrame = new JFrame("Java SWING Examples");
     mainFrame.setSize(400,400);
     mainFrame.setLayout(new GridLayout(3, 1));

     headerLabel = new JLabel("",JLabel.CENTER );
     statusLabel = new JLabel("",JLabel.CENTER);
     statusLabel.setSize(350,100);

     mainFrame.addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent windowEvent){
         System.exit(0);
}
});
     controlPanel = new JPanel();
     controlPanel.setLayout(new FlowLayout());

     mainFrame.add(headerLabel);
     mainFrame.add(controlPanel);
     mainFrame.add(statusLabel);
     mainFrame.setVisible(true);
   }
   private void showEventDemo(){
     headerLabel.setText("Control in action: Button");

     JButton okButton = new JButton("OK");
     JButton submitButton = new JButton("Submit");
     JButton cancelButton = new JButton("Cancel");
```

```
    okButton.setActionCommand("OK");
    submitButton.setActionCommand("Submit");
    cancelButton.setActionCommand("Cancel");

    okButton.addActionListener(new                    ButtonClickListener());
    submitButton.addActionListener(new                ButtonClickListener());
    cancelButton.addActionListener(new ButtonClickListener());

    controlPanel.add(okButton);
    controlPanel.add(submitButton);
    controlPanel.add(cancelButton);

    mainFrame.setVisible(true);
  }
  private class ButtonClickListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
      String command = e.getActionCommand();

      if( command.equals( "OK" )) {
        statusLabel.setText("Ok Button clicked.");
      } else if( command.equals( "Submit" ) ) {
        statusLabel.setText("Submit Button clicked.");
      } else {
        statusLabel.setText("Cancel Button clicked.");
      }
    }
  }
}
```

Compile the program using the command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\AWT>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error occurs, it means the compilation is successful. Run the program using the following command.

```
D:\AWT>java  com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output.

JavaScript | Output

JavaScript Output defines the ways to display the output of a given code. The output can be display by using four different ways which are listed below:

- **innerHTML:** It is used to access an element. It defines the HTML content.
  Syntax:

  ```
  document.getElementById(id)
  ```

  Example: This example uses innerHTML to display the data.

  filter_none

  edit
  play_arrow
  brightness_4

  ```
  <!DOCTYPE html>

  <html>
  ```

```
<head>
    <title>
        JavaScript Output using innerHTML
    </title>
</head>

<body>
    <h1>GeeksforGeeks</h1>

    <h2>
        JavaScript     Display     Possibilities
        Using innerHTML
    </h2>

    <p id="GFG"></p>

    <!-- Script to uses innerHTML -->
    <script>
        document.getElementById("GFG").innerHTML
                = 10 * 2;
    </script>
```

```
</body>


</html>
```

Output:

- **document.write():** It is used for testing purpose.

Syntax:

```
document.write()
```

Example: This example uses document.write() property to display data.

filter_none

edit
play_arrow
brightness_4

```
<!DOCTYPE html>

<html>


<head>

    <title>

        JavaScript Output using document.write()

    </title>

</head>


<body>

    <h1>GeeksforGeeks</h1>


    <h2>
```

```
        JavaScript Display Possibilities
        Using document.write()
    </h2>

    <p id="GFG"></p>

    <!-- Script to uses document.write() -->
    <script>
        document.write(10 * 2);
    </script>
</body>

</html>
```

Output:

- **window.alert():**It displays the content using an alert box.

Syntax:

```
window.alert()
```

Example: This example uses window.alert() property to display data.

filter_none

edit
play_arrow
brightness_4

```
<!DOCTYPE html>

<html>
```

```
<head>
    <title>
        JavaScript Output using window.alert()
    </title>
</head>

<body>
    <h1>GeeksforGeeks</h1>

    <h2>
        JavaScript Display Possibilities
        Using window.alert()
    </h2>

    <p id="GFG"></p>

    <!-- Script to use window.alert() -->
    <script>
        window.alert(10 * 2);
    </script>
</body>
```

```
</html>
```

Output:

- **console.log():** It is used for debugging purposes.

Syntax:

```
console.log()
```

Example: This example uses console.log() property to display data.

filter_none

edit
play_arrow
brightness_4

```
<!DOCTYPE html>

<html>


<head>

    <title>

        JavaScript Output using innerHTML

    </title>

</head>


<body>

    <h1>GeeksforGeeks</h1>


    <h2>

        JavaScript Display Possibilities
```

```
        Using console.log()
    </h2>

    <p id="GFG"></p>

    <!-- Script to use console.log() -->
    <script>
        console.log(10*2);
    </script>
</body>

</html>
```

JSP - Form Handling

The Methods in Form Processing

Let us now discuss the methods in Form Processing.

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 ? character as follows −

http://www.test.com/hello?key1=value1&key2=value2

The GET method is the default method to pass information from the browser to the web server and it produces a long string that appears in your

browser's **Location:box**. It is recommended that the GET method is better not used. if you have password or other sensitive information to pass to the server.

The GET method has size limitation: **only 1024 characters can be in a request string**.

This information is passed using **QUERY_STRING header** and will be accessible through QUERY_STRING environment variable which can be handled using **getQueryString()** and **getParameter()** methods of request object.

POST method

A generally more reliable method of passing information to a backend program is the POST method.

This method packages the information in exactly the same way as the GET method, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing.

JSP handles this type of requests using **getParameter()** method to read simple parameters and **getInputStream()** method to read binary data stream coming from the client.

Reading Form Data using JSP

JSP 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.
- **getInputStream()** − Call this method to read binary data stream coming from the client.

GET Method Example Using URL

The following URL will pass two values to HelloForm program using the GET method.

http://localhost:8080/main.jsp?first_name=ZARA&last_name=ALI

Below is the **main.jsp** JSP program to handle input given by web browser. We are going to use the **getParameter()** method which makes it very easy to access the passed information −

```
<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>
```

Now type **http://localhost:8080/main.jsp?first_name=ZARA&last_name=ALI** in your browser's **Location:box**. This will generate the following result −

Using GET Method to Read Form Data

- **First Name**: ZARA
- **Last Name**: ALI

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>
```

Keep this HTML in a file Hello.htm and put it in **<Tomcat-installation-directory>/webapps/ROOT directory**. When you would access **http://localhost:8080/Hello.htm**, you will receive the following output.

First Name:

Last Name:

< p>Try to enter the First Name and the Last Name and then click the 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 a little modification in the above JSP to handle both the GET and the POST method. Below is the **main.jsp** JSP program to handle the input given by web browser using the GET or the POST methods.

Infact there is no change in the above JSP because the only way of passing parameters is changed and no binary data is being passed to the JSP program. File handling related concepts will be explained in separate chapter where we need to read the binary data stream.

```
<html>
  <head>
    <title>Using GET and POST Method to Read Form Data</title>
  </head>

  <body>
    <center>
    <h1>Using POST 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>
```

Following is the content of the **Hello.htm** file −

```
<html>
  <body>

    <form action = "main.jsp" 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>
```

Let us now keep **main.jsp** and hello.htm in **<Tomcat-installationdirectory>/webapps/ROOT directory**. When you access **http://localhost:8080/Hello.htm**, you will receive the following output.

First Name:

Last Name:

Try to enter the First and the Last Name and then click the submit button to see the result on your local machine where tomcat is running.

Based on the input provided, you will receive similar results as in the above examples.

Passing Checkbox Data to JSP Program

Checkboxes are used when more than one option is required to be selected.

Following is an example **HTML code, CheckBox.htm**, for a form with two checkboxes.

```
<html>
  <body>

    <form action = "main.jsp" method = "POST" target = "_blank">
      <input type = "checkbox" name = "maths" checked = "checked" /> Maths
      <input type = "checkbox" name = "physics" /> Physics
      <input type = "checkbox" name = "chemistry" checked = "checked" />
Chemistry
      <input type = "submit" value = "Select Subject" />
    </form>

  </body>
</html>
```

The above code will generate the following result −

☑ Maths ☐ Physics ☑ Chemistry

Following is main.jsp JSP program to handle the input given by the web browser for the checkbox button.

```
<html>
  <head>
    <title>Reading Checkbox Data</title>
  </head>

  <body>
    <h1>Reading Checkbox Data</h1>

    <ul>
      <li><p><b>Maths Flag:</b>
        <%= request.getParameter("maths")%>
      </p></li>
      <li><p><b>Physics Flag:</b>
        <%= request.getParameter("physics")%>
      </p></li>
      <li><p><b>Chemistry Flag:</b>
```

```
         <%= request.getParameter("chemistry")%>
       </p></li>
     </ul>

   </body>
</html>
```

The above program will generate the following result −

Reading Checkbox Data

- **Maths Flag :**: on
- **Physics Flag:**: null
- **Chemistry Flag:**: on

Reading All Form Parameters

Following is a generic example which uses **getParameterNames()** method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order.

Once we have an Enumeration, we can loop down the Enumeration in the standard manner, using the **hasMoreElements()** method to determine when to stop and using the **nextElement()** method to get each parameter name.

```
<%@ page import = "java.io.*,java.util.*" %>

<html>
  <head>
    <title>HTTP Header Request Example</title>
  </head>

  <body>
    <center>
      <h2>HTTP Header Request Example</h2>
```

```
      <table width = "100%" border = "1" align = "center">
        <tr bgcolor = "#949494">
          <th>Param Name</th>
          <th>Param Value(s)</th>
        </tr>
        <%
          Enumeration paramNames = request.getParameterNames();
          while(paramNames.hasMoreElements()) {
            String paramName = (String)paramNames.nextElement();
            out.print("<tr><td>" + paramName + "</td>\n");
            String paramValue = request.getHeader(paramName);
            out.println("<td> " + paramValue + "</td></tr>\n");
          }
        %>
      </table>
    </center>

  </body>
</html>
```

Following is the content of the **Hello.htm** −

```
<html>
  <body>

    <form action = "main.jsp" method = "POST" target = "_blank">
      <input type = "checkbox" name = "maths" checked = "checked" /> Maths
      <input type = "checkbox" name = "physics" /> Physics
      <input type = "checkbox" name = "chemistry" checked = "checked" /> Chem
      <input type = "submit" value = "Select Subject" />
    </form>

  </body>
</html>
```

Now try calling JSP using the above Hello.htm; this would generate a result something like as below based on the provided input −

Reading All Form Parameters

Param Name	Param Value(s)

Maths	on
Chemistry	on

You can try the above JSP to read any other form's data which is having other objects like text box, radio button or dropdown, etc.

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). A servlet that sets a cookie might send headers that look something like this −

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name = xyz; expires = Friday, 04-Feb-07 22:03:38 GMT;
   path = /; domain = tutorialspoint.com
Connection:          close
Content-Type: text/html
```

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this −

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name = xyz
```

A servlet will then have access to the cookie through the request method request.getCookies() which returns an array of Cookie objects.

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.

Setting Cookies with Servlet

Setting cookies with servlet involves three steps −

(1) Creating a Cookie object − You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

```
Cookie cookie = new Cookie("key","value");
```

Keep in mind, neither the name nor the value should contain white space or any of the following characters −

```
[ ] ( ) = , " / ? @ : ;
```

(2) Setting the maximum age − You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

```
cookie.setMaxAge(60 * 60 * 24);
```

(3) Sending the Cookie into the HTTP response headers − You use response.addCookie to add cookies in the HTTP response header as follows −

```
response.addCookie(cookie);
```

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>
```

Keep above HTML content 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. This would display first name and last name on your screen and same time it would set two cookies firstName and lastName which would be passed back to the server when next time you would press Submit button.

Next section would explain you how you would access these cookies back in your web application.

Reading Cookies with Servlet

To read cookies, you need to create an array of javax.servlet.http.Cookie objects by calling the **getCookies()** method of HttpServletRequest. Then cycle through the array, and use getName() and getValue() methods to access each cookie and associated value.

Example

Let us read cookies which we have set in previous example −

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class ReadCookies extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    Cookie cookie = null;
    Cookie[] cookies = null;

    // Get an array of Cookies associated with this domain
    cookies = request.getCookies();

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Reading 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" );

    if( cookies != null ) {
      out.println("<h2> Found Cookies Name and Value</h2>");

      for (int i = 0; i < cookies.length; i++) {
        cookie = cookies[i];
        out.print("Name : " + cookie.getName( ) + ", ");
```

```
            out.print("Value: " + cookie.getValue( ) + " <br/>");
         }
      } else {
         out.println("<h2>No cookies founds</h2>");
      }
      out.println("</body>");
      out.println("</html>");
   }
}
```

Compile above servlet **ReadCookies** and create appropriate entry in web.xml file. If you would have set first_name cookie as "John" and last_name cookie as "Player" then running http://localhost:8080/ReadCookies would display the following result −

Found Cookies Name and Value

Name : first_name, Value: John

Name : last_name, Value: Player

Delete Cookies with Servlet

To delete cookies is very simple. If you want to delete a cookie then you simply need to follow up following three steps −

- Read an already existing cookie and store it in Cookie object.
- Set cookie age as zero using **setMaxAge()** method to delete an existing cookie
- Add this cookie back into response header.

Example

The following example would delete and existing cookie named "first_name" and when you would run ReadCookies servlet next time it would return null value for first_name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
// Extend HttpServlet class
public class DeleteCookies extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    Cookie cookie = null;
    Cookie[] cookies = null;

    // Get an array of Cookies associated with this domain
    cookies = request.getCookies();

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Delete 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" );

    if( cookies != null ) {
      out.println("<h2> Cookies Name and Value</h2>");

      for (int i = 0; i < cookies.length; i++) {
        cookie = cookies[i];

        if((cookie.getName( )).compareTo("first_name") == 0 ) {
          cookie.setMaxAge(0);
          response.addCookie(cookie);
          out.print("Deleted cookie : " + cookie.getName( ) + "<br/>");
        }
        out.print("Name : " + cookie.getName( ) + ", ");
        out.print("Value: " + cookie.getValue( )+" <br/>");
      }
```

```
    } else {
      out.println("<h2>No cookies founds</h2>");
    }
    out.println("</body>");
    out.println("</html>");
  }
}
```

Compile above servlet **DeleteCookies** and create appropriate entry in web.xml file. Now running http://localhost:8080/DeleteCookies would display the following result −

Cookies Name and Value

Deleted cookie : first_name

Name : first_name, Value: John

Name : last_name, Value: Player

Now try to run http://localhost:8080/ReadCookies and it would display only one cookie as follows −

Found Cookies Name and Value

Name : last_name, Value: Player

You can delete your cookies in Internet Explorer manually. Start at the Tools menu and select Internet Options. To delete all cookies, press Delete Cookies.

Hidden fields and Objects

HTML DOM Input Hidden Object

Input Hidden Object

The Input Hidden object represents an HTML <input> element with type="hidden".

Access an Input Hidden Object

You can access an <input> element with type="hidden" by using getElementById():

Example

```
var x = document.getElementById("myInput");
```

Tip: You can also access <input type="hidden"> by searching through the elements collection of a form.

Create an Input Hidden Object

You can create an <input> element with type="hidden" by using the document.createElement() method:

Example

```
var x = document.createElement("INPUT");
x.setAttribute("type", "hidden");
```

Input Hidden Object Properties

Property	Description
defaultValue	Sets or returns the default value of the hidden input field

form	Returns a reference to the form that contains the hidden input field
name	Sets or returns the value of the name attribute of the hidden input field
type	Returns which type of form element a hidden input field is
value	Sets or returns the value of the value attribute of the hidden input field

HTML DOM Input Image Object

Input Image Object

The Input Image object represents an HTML <input> element with type="image".

Access an Input Image Object

You can access an <input> element with type="image" by using getElementById():

```
var x = document.getElementById("myImage");
```

Tip: You can also access <input type="image"> by searching through the elements collection of a form.

Create an Input Image Object

You can create an <input> element with type="image" by using the document.createElement() method:

```
var x = document.createElement("INPUT");
x.setAttribute("type", "image");
```

Input Image Object Properties

Property	Description
Alt	Sets or returns the value of the alt attribute of an input image
Autofocus	Sets or returns whether an input image should automatically get focus when the page loads
defaultValue	Sets or returns the default value of an input image
Disabled	Sets or returns whether an input image is disabled, or not
Form	Returns a reference to the form that contains the input image
formAction	Sets or returns the value of the formaction attribute of an input image
formEnctype	Sets or returns the value of the formenctype attribute of an input image
formMethod	Sets or returns the value of the formmethod attribute of an input image
formNoValidate	Sets or returns whether the form-data should be validated or not, on submission
formTarget	Sets or returns the value of the formtarget attribute an input image

Height	Sets or returns the value of the height attribute of the input image
Name	Sets or returns the value of the name attribute of an input image
Src	Sets or returns the value of the src attribute of the input image
Type	Returns which type of form element the input image is
Value	Sets or returns the value of the value attribute of an input image
Width	Sets or returns the value of the width attribute of the input image

Java language was developed by Sun Microsystems in 1995. In subsequent years, the language has become the backbone of millions of applications across multiple platforms including Windows, Macintosh and UNIX-based desktops, Android-based mobiles, embedded systems and enterprise solutions. According to Oracle (that acquired Sun Microsystems in 2010), Java now runs on more than 3 billion devices.

Types of Applications that Run on Java

1. Desktop GUI Applications:

Java provides GUI development through various means like Abstract Windowing Toolkit (AWT), Swing and JavaFX. While AWT contains a number of pre-constructed components such as menu, button, list, and numerous third-party components, Swing, a GUI widget toolkit, additionally provides certain advanced components like trees, tables, scroll panes, tabbed panel and lists. JavaFX, a set of

graphics and media packages, provides Swing interoperability, 3D graphic features and self-contained deployment model which facilitates quick scripting of Java applets and applications.

2. Mobile Applications:

Java Platform, Micro Edition (Java ME or J2ME) is a cross-platform framework to build applications that run across all Java supported devices, including feature phones and smart phones. Further, applications for Android, one of the most popular mobile operating systems, are usually scripted in Java using the Android Software Development Kit (SDK) or other environments.

3. Embedded Systems:

Embedded systems, ranging from tiny chips to specialized computers, are components of larger electromechanical systems performing dedicated tasks. Several devices, such as SIM cards, blue-ray disk players, utility meters and televisions, use embedded Java technologies. According to Oracle, 100% of Blu-ray Disc Players and 125 million TV devices employ Java.

4. Web Applications:

Java provides support for web applications through Servlets, Struts or JSPs. The easy programming and higher security offered by the programming language has allowed a large number of government applications for health, social security, education and insurance to be based on Java. Java also finds application in development of eCommerce web applications using open-source eCommerce platforms, such as Broadleaf.

5. Web Servers and Application Servers:

The Java ecosystem today contains multiple Java web servers and application servers. While Apache Tomcat, Simple, Jo!, Rimfaxe Web Server (RWS) and Project Jigsaw dominate the web server space, WebLogic, WebSphere, and Jboss EAP dominate commercial application server space.

6. Enterprise Applications:

Java Enterprise Edition (Java EE) is a popular platform that provides API and runtime environment for scripting and running enterprise software, including

network applications and web-services. Oracle claims Java is running in 97% of enterprise computers. The higher performance guarantee and faster computing in Java has resulted in high frequency trading systems like Murex to be scripted in the language. It is also the backbone for a variety of banking applications which have Java running from front user end to back server end.

7. Scientific Applications:

Java is the choice of many software developers for writing applications involving scientific calculations and mathematical operations. These programs are generally considered to be fast and secure, have a higher degree of portability and low maintenance. Applications like MATLAB use Java both for interacting user interface and as part of the core system.

UNIT-3

What is Java Server Pages?

Java Server Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A Java Server Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the **Common Gateway Interface (CGI)**. But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including **JDBC, JNDI, EJB, JAXP,** etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

Advantages of JSP

Following table lists out the other advantages of using JSP over other technologies –

vs. Active Server Pages (ASP)

The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

vs. Pure Servlets

It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

vs. Server-Side Includes (SSI)

SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

vs. JavaScript

JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

vs. Static HTML

Regular HTML, of course, cannot contain dynamic information.

Common Gateway Interface

Common Gateway Interface (CGI)

The **Common Gateway Interface (CGI)** provides the middleware between WWW servers and external databases and information sources. The World Wide Web Consortium (W3C) defined the Common Gateway Interface (CGI) and also

defined how a program interacts with a Hyper Text Transfer Protocol (HTTP) server. The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message. This process or convention for passing data back and forth between the server and the application is called the common gateway interface (CGI).

Features of CGI:

- It is a very well defined and supported standard.
- CGI scripts are generally written in either Perl, C, or maybe just a simple shell script.
- CGI is a technology that interfaces with HTML.
- CGI is the best method to create a counter because it is currently the quickest
- CGI standard is generally the most compatible with today's browsers

Advantages of CGI:

- The advanced tasks are currently a lot easier to perform in CGI than in Java.
- It is always easier to use the code already written than to write your own.
- CGI specifies that the programs can be written in any language, and on any platform, as long as they conform to the specification.
- CGI-based counters and CGI code to perform simple tasks are available in plenty.

Disadvantages of CGI:

There are some disadvantages of CGI which are given below:

- In Common Gateway Interface each page load incurs overhead by having to load the programs into memory.
- Generally, data cannot be easily cached in memory between page loads.
- There is a huge existing code base, much of it in Perl.
- CGI uses up a lot of processing time.

ASP

ASP (Active Server Page) is a Microsoft technology that has been widely used by many web businesses. Similar to JSP, PHP or CGI, it enables a dynamic HTML page to be displayed from the server when the page is requested from the client browser. With ASP, developers can combine HTML pages, script commands and COM components to generate dynamic and interactive web server application.

ASP can be developed using default scripting language, VBScript, or any language that support ActiveX scripting.

ASP has been integrated with the Microsoft IIS (Internet Information Server) platform. The ASP engine is started on the web server when a file with an .asp extension is used. This engine resides in a DLL file called asp.dll that runs in memory along with IIS. You can develop ASP pages through Microsoft FrontPage and test your ASP pages on IIS running on a local Windows NT or Windows 2000 workstation before you upload them to the server.

ASP .NET is a Microsoft programming framework that built on the common language runtime on the web server to build powerful web applications. Both ASP and ASP .NET applications can run together on a server independently because they have different file extension (.asp and .aspx) and different configuration models (registry versus XML-based configuration files). In additional, these two systems have totally separate processing engines.

Since ASP and ASP .NET are Microsoft technology, you will need to find a web hosting provider that support Microsoft Window Server in order to host your website created in ASP .NET or ASP. But, you can run ASP applications on Linux-based server environment too. Application such as " Sun ChiliSoft ASP for Linux" if installed on the Linux server will be able to support ASP application.

WWW and I/O operations on it

Overview

WWW stands for **World Wide Web.** A technical definition of the World Wide Web is : all the resources and users on the Internet that are using the Hypertext Transfer Protocol (HTTP).

A broader definition comes from the organization that Web inventor **Tim Berners-Lee** helped found, the **World Wide Web Consortium (W3C).**

The World Wide Web is the universe of network-accessible information, an embodiment of human knowledge.

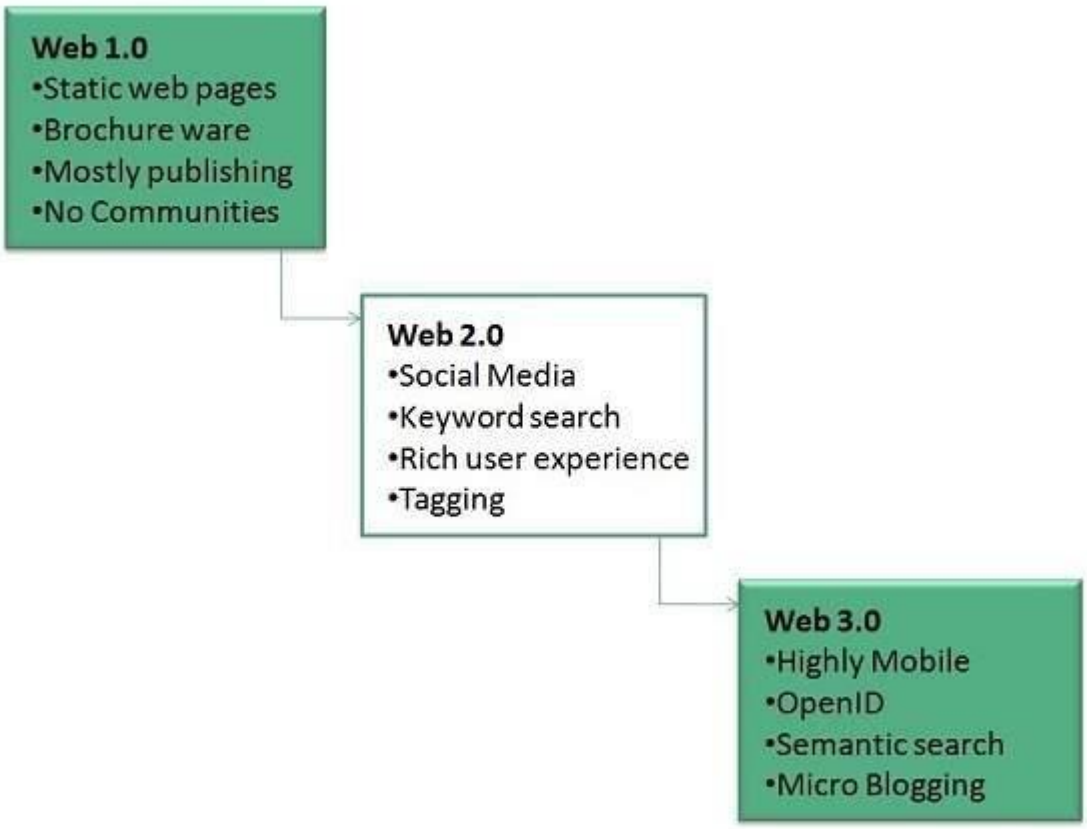
In simple terms, The World Wide Web is a way of exchanging information between computers on the Internet, tying them together into a vast collection of interactive multimedia resources.

Internet and **Web** is not the same thing: Web uses internet to pass over the information.

Evolution

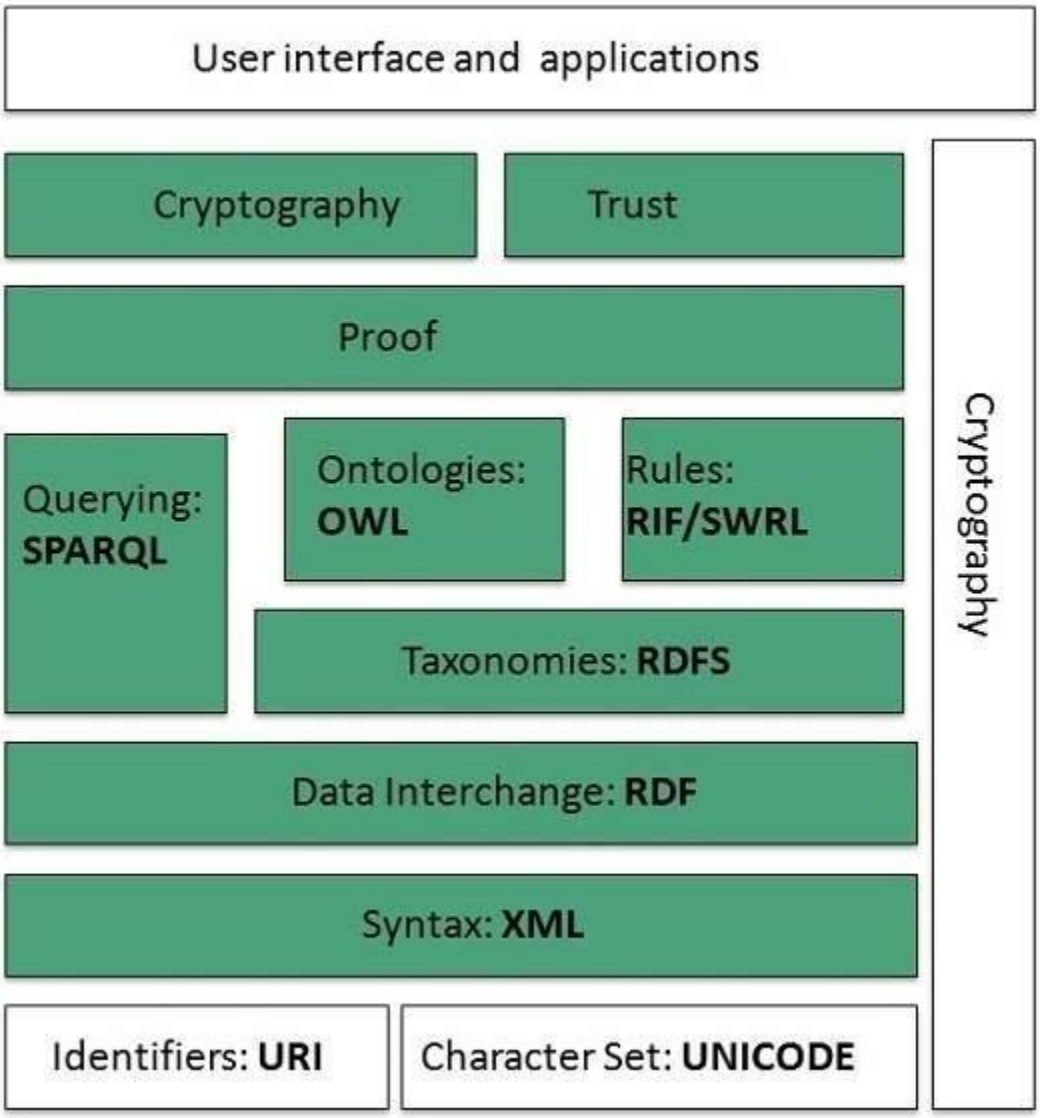
World Wide Web was created by **Timothy Berners Lee** in 1989 at **CERN** in **Geneva.** World Wide Web came into existence as a proposal by him, to allow researchers to work together effectively and efficiently at **CERN.** Eventually it became **World Wide Web.**

The following diagram briefly defines evolution of World Wide Web:



WWW Architecture

WWW architecture is divided into several layers as shown in the following diagram:



Identifiers and Character Set

Uniform Resource Identifier (URI) is used to uniquely identify resources on the web and **UNICODE** makes it possible to built web pages that can be read and write in human languages.

Syntax

XML (Extensible Markup Language) helps to define common syntax in semantic web.

Data Interchange

Resource Description Framework (RDF) framework helps in defining core representation of data for web. RDF represents data about resource in graph form.

Taxonomies

RDF Schema (RDFS) allows more standardized description of **taxonomies** and other **ontological** constructs.

Ontologies

Web Ontology Language (OWL) offers more constructs over RDFS. It comes in following three versions:

- OWL Lite for taxonomies and simple constraints.
- OWL DL for full description logic support.
- OWL for more syntactic freedom of RDF

Rules

RIF and **SWRL** offers rules beyond the constructs that are available from **RDFs** and **OWL.** Simple Protocol and **RDF Query Language (SPARQL)** is SQL like language used for querying RDF data and OWL Ontologies.

Proof

All semantic and rules that are executed at layers below Proof and their result will be used to prove deductions.

Cryptography

Cryptography means such as digital signature for verification of the origin of sources is used.

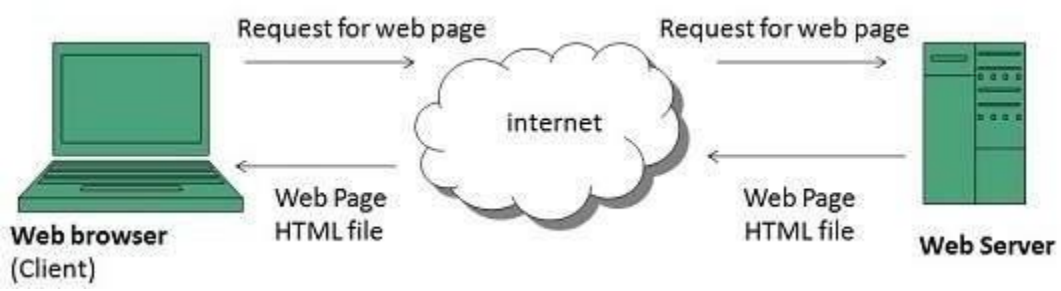
User Interface and Applications

On the top of layer **User interface and Applications** layer is built for user interaction.

WWW Operation

WWW works on client- server approach. Following steps explains how the web works:

1. User enters the URL (say, **http://www.tutorialspoint.com**) of the web page in the address bar of web browser.
2. Then browser requests the Domain Name Server for the IP address corresponding to www.tutorialspoint.com.
3. After receiving IP address, browser sends the request for web page to the web server using HTTP protocol which specifies the way the browser and web server communicates.
4. Then web server receives request using HTTP protocol and checks its search for the requested web page. If found it returns it back to the web browser and close the HTTP connection.
5. Now the web browser receives the web page, It interprets it and display the contents of web page in web browser's window.



Future

There had been a rapid development in field of web. It has its impact in almost every area such as education, research, technology, commerce, marketing etc. So the future of web is almost unpredictable.

Apart from huge development in field of WWW, there are also some technical issues that W3 consortium has to cope up with.

User Interface

Work on higher quality presentation of 3-D information is under deveopment. The W3 Consortium is also looking forward to enhance the web to full fill requirements of global communities which would include all regional languages and writing systems.

Technology

Work on privacy and security is under way. This would include hiding information, accounting, access control, integrity and risk management.

Architecture

There has been huge growth in field of web which may lead to overload the internet and degrade its performance. Hence more better protocol are required to be developed.

Form validation using HTML and JavaScript

Forms are used in webpages for the user to enter their required details that are further send it to the server for processing. A form is also known as web form or HTML form. Examples of form use are prevalent in e-commerce websites, online banking, online surveys to name a few

Syntax for form in HTML

```
<body>

<h1 style="text-align: center"> REGISTRATION FORM </h1>

<form name="RegForm" action="/submit.php" onsubmit="return
GEEKFORGEEKS()" method="post">


   <p>Name: <input type="text" size=65 name="Name"> </p><br>

   <p> Address: <input type="text" size=65 name="Address"> </p><br>

   <p>E-mail Address: <input type="text" size=65 name="EMail"> </p><br>

    <p>Password: <input type="text" size=65 name="Password"> </p><br>

   <p>Telephone: <input type="text" size=65 name="Telephone"> </p><br>
```

```
    <p>SELECT YOUR COURSE
        <select type="text" value="" name="Subject">
            <option>BTECH</option>
            <option>BBA</option>
            <option>BCA</option>
            <option>B.COM</option>
            <option>GEEKFORGEEKS</option>
        </select></p><br><br>
    <p>Comments: <textarea cols="55" name="Comment"> </textarea></p>
    <p><input type="submit" value="send" name="Submit">
        <input type="reset" value="Reset" name="Reset">
    </p>
</form>
</body>
```

Validating a form :

The data entered into a form needs to be in the right format and certain fields need to be filled in order to effectively use the submitted form. Username, password, contact information are some details that are mandatory in forms and thus need to be provided by the user.

Form validation using HTML and JavaScript

Forms are used in webpages for the user to enter their required details that are further send it to the server for processing. A form is also known as web form or HTML form. Examples of form use are prevalent in e-commerce websites, online banking, online surveys to name a few

Syntax for form in HTML

```
<body>

<h1 style="text-align: center"> REGISTRATION FORM </h1>

<form name="RegForm" action="/submit.php" onsubmit="return
GEEKFORGEEKS()" method="post">


    <p>Name: <input type="text" size=65 name="Name"> </p><br>

    <p> Address: <input type="text" size=65 name="Address"> </p><br>

    <p>E-mail Address: <input type="text" size=65 name="EMail"> </p><br>

     <p>Password: <input type="text" size=65 name="Password"> </p><br>

    <p>Telephone: <input type="text" size=65 name="Telephone"> </p><br>


    <p>SELECT YOUR COURSE

        <select type="text" value="" name="Subject">

            <option>BTECH</option>

            <option>BBA</option>

            <option>BCA</option>

            <option>B.COM</option>

            <option>GEEKFORGEEKS</option>

        </select></p><br><br>

    <p>Comments: <textarea cols="55" name="Comment"> </textarea></p>

    <p><input type="submit" value="send" name="Submit">

        <input type="reset" value="Reset" name="Reset">

    </p>
```

```
</form>

</body>
```

Validating a form :
The data entered into a form needs to be in the right format and certain fields need to be filled in order to effectively use the submitted form. Username, password, contact information are some details that are mandatory in forms and thus need to be provided by the user.

Below is a code in HTML, CSS and JavaScript to validate a form .
HTML is used to create the form.
JavaScript to validate the form.
CSS to design the layout of the form.
Form validation :

```
<script>

function GEEKFORGEEKS()

{

    var  name  =  document.forms["RegForm"]["Name"];

    var email = document.forms["RegForm"]["EMail"];

    var  phone  =  document.forms["RegForm"]["Telephone"];

    var what =  document.forms["RegForm"]["Subject"];

    var  password  =  document.forms["RegForm"]["Password"];

    var address = document.forms["RegForm"]["Address"];


    if (name.value == "")

    {
```

```
    window.alert("Please enter your name.");
    name.focus();
    return false;
}

if (address.value == "")
{
    window.alert("Please enter your address.");
    address.focus();
    return false;
}

if (email.value == "")
{
    window.alert("Please enter a valid e-mail address.");
    email.focus();
    return false;
}

if (phone.value == "")
{
    window.alert("Please enter your telephone number.");
```

```
        phone.focus();
        return false;
    }

    if (password.value == "")
    {
        window.alert("Please enter your password");
        password.focus();
        return false;
    }

    if (what.selectedIndex < 1)
    {
        alert("Please enter your course.");
        what.focus();
        return false;
    }

    return true;
}</script>
```

Styling the form :

```
internet {
    margin-left:  70px;
    font-weight: bold ;
    float: left;
    clear:        left;
    width:   100px;
    text-align: left;
    margin-right: 10px;
    font-family:sans-serif,bold, Arial, Helvetica;
    font-size:14px;
}

div {
    box-sizing:        border-box;
    width: 100%;
    border:   100px   solid   black;
    float: left;
    align-content:  center;
    align-items: center;
}

form {
```

```
    margin:  0  auto;

    width: 600px;

}</style>
```

Form Processing using Perl

Form Processing Program process.pl

This program reads in the data provided by form.html and creates a web page using that information. The program is written in perl, and must be placed in the cgi-bin directory. The permissions on the file must be set properly so that the program is executable.

The program uses a CGI Perl module that has certain functions, one of which is **&ReadParse**. That function grabs the data that was passed by the form, and puts it into something called an associative array, also known as a **hash**. The hash, **%in**, can allow the new web page to find the values of the passed variables by referencing keys relating to the variable names. The command **$in{ $key }** would return the value of the variable associated with that key.

To print out the web page itself, much of the standard HTML code looks the same, but is actually encased in a long **print** statement. In perl, the print statement can print multiple lines until it reaches some label. Things that require computation in perl are located outside the print statements.

```
#!/usr/bin/perl

# Program Name: process.pl
# Author: D. W. Hyatt
# This program uses the data passed by the form program and creates
# a new web page with that information.

use CGI qw(:cgi-lib :standard); # Use CGI modules that let people read data passed from a form


&ReadParse(%in);                # This grabs the data passed by the form and puts it in an array
```

```
$name = $in{"name"};            # Get the user's name and assign to variable
$preference = $in{"choice"};    # Get the choice and assign to variable

                    # Start printing HTML document
print<<EOSTUFF;
Content-type: text/html

<HTML>
<BODY BGCOLOR=WHITE TEXT=BLACK>
<H1> Hello, $name </H1>         <!-- Use variables in HTML text -->
You prefer  $preference.
<BR>
EOSTUFF

for ($i=1; $i<=5; $i++)         # Print name 5 times
 {print "$i. $name <BR>";}

print<<EOF;                 <!-- Finish up document -->
</BODY>
</HTML>
EOF
```

VBScript and Forms

Simple Validation

You can use Visual Basic Scripting Edition to do much of the form processing that you'd usually have to do on a server. You can also do things that just can't be done on the server.

Here's an example of simple client-side validation. The HTML code is for a text box and a button. If you use Microsoft® Internet Explorer to view the page produced by the following code, you'll see a small text box with a button next to it.

```
<HTML>
<HEAD><TITLE>Simple Validation</TITLE>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub     Submit_OnClick
  Dim TheForm
  Set TheForm = Document.ValidForm
  If IsNumeric(TheForm.Text1.Value) Then
    If  TheForm.Text1.Value  <  1  Or  TheForm.Text1.Value  >  10  Then
      MsgBox "Please enter a number between 1 and 10."
    Else
      MsgBox  "Thank  you."
    End If
  Else
    MsgBox  "Please  enter  a  numeric  value."
  End If
End Sub
-->
</SCRIPT>
</HEAD>
<BODY>
<H3>Simple Validation</H3><HR>
<FORM        NAME="ValidForm">
Enter a value between 1 and 10:
<INPUT NAME="Text1" TYPE="TEXT" SIZE="2">
<INPUT NAME="Submit" TYPE="BUTTON" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

The difference between this text box and the examples on A Simple VBScript Page is that the **Value** property of the text box is used to check the entered value. To get the **Value** property, the code has to qualify the reference to the name of the text box.

You can always write out the full reference Document.ValidForm.Text1. However, where you have multiple references to form controls, you'll want to do what was done here. First declare a variable. Then use the **Set** statement to assign the form to the variable TheForm. A regular assignment statement, such as **Dim**, doesn't work here; you must use **Set** to preserve the reference to an object.

Using Numeric Values

Notice that the example directly tests the value against a number: it uses the **IsNumeric** function to make sure the string in the text box is a number. Although VBScript automatically converts strings and numbers, it's always a good practice to test a user-entered value for its data subtype and to use conversion functions as necessary. When doing addition with text box values, convert the values explicitly to numbers because the plus sign (+) operator represents both addition and string concatenation. For example, if Text1 contains "1" and Text2 contains "2", you see the following results:

```
A = Text1.Value + Text2.Value              ' A is "12"
A = CDbl(Text1.Value) + Text2.Value   ' A is 3
```

Validating and Passing Data Back to the Server

The simple validation example uses a plain button control. If a Submit control was used, the example would never see the data to check it—everything would go immediately to the server. Avoiding the Submit control lets you check the data, but it doesn't submit the data to the server. That requires an additional line of code:

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub     Submit_OnClick
  Dim TheForm
  Set TheForm = Document.ValidForm
  If IsNumeric(TheForm.Text1.Value) Then
    If  TheForm.Text1.Value  <  1  Or  TheForm.Text1.Value  >  10  Then
      MsgBox "Please enter a number between 1 and 10."
    Else
      MsgBox "Thank you."
      TheForm.Submit      ' Data correct; send to server.
    End If
  Else
```

```
    MsgBox "Please enter a numeric value."
  End If
End Sub
-->
</SCRIPT>
```

To send the data to the server, the code invokes the **Submit** method on the form object when the data is correct. From there, the server handles the data just as it otherwise would—except that the data is correct before it gets there. You'll find complete information about the **Submit** method and other methods on the Internet Explorer Scripting Object Model page.

So far, you've seen only the standard HTML <FORM> objects. Internet Explorer also lets you exploit the full power of ActiveX™ controls (formerly called OLE controls) and Java™ objects.

Unit -4

INTRODUCTION TO ASP/JSP- already discussed in unit-3

VRML Idea:- **VRML (Virtual Reality Modeling Language)**

VRML (Virtual Reality Modeling Language) is a language for describing three-dimensional (3-D) image sequences and possible user interactions to go with them. Using VRML, you can build a sequence of visual images into Web settings with which a user can interact by viewing, moving, rotating, and otherwise interacting with an apparently 3-D scene. For example, you can view a room and use controls to move the room as you would experience it if you were walking through it in real space.

To view a VRML file, you need a VRML viewer or browser, which can be a plug-in for a Web browser you already have. Among viewers you can download for the Windows platforms are blaxxun's CC Pro, Platinum's Cosmo Player, WebFX, WorldView, and Fountain. Whurlwind and Voyager are two viewers for the Mac.

How do I open a VRML file?

To view a **VRML** model that is stored on your local disk, choose "**Open**..." from the **File** menu. Click on "Browse" from the **Open** dialog that appears. Change the "**Files** of type" selector to "All **Files**", then **open** the file.

What is VRML how does it differ from HTML?

VRML is to 3D what **HTML** is to 2D. While **HTML** specifies how two-dimensional documents are represented, **VRML** is a format that describes how three-dimensional environments can be explored and created on the World Wide Web.

What are nodes in VRML?

VRML Nodes. A **VRML** world is just a set of **nodes**. Each **node** will have a type, scale, rotation, translation, color, texture, and some others. There will often be useful defaults for all these properties. **Nodes** have for basic types: infomational, grouping, shape, and transformational.

Java Applet

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following −

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet

Four methods in the Applet class gives you the framework on which you build any serious applet −

- **init** − This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start** − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

A "Hello, World" Applet

Following is a simple applet named HelloWorldApplet.java −

```
import java.applet.*;
import java.awt.*;

public class HelloWorldApplet extends Applet {
   public void paint (Graphics g) {
      g.drawString ("Hello World", 25, 50);
   }
}
```

These import statements bring the classes into the scope of our applet class −

- java.applet.Applet
- java.awt.Graphics

Without those import statements, the Java compiler would not recognize the classes Applet and Graphics, which the applet class refers to.

The Applet Class

Every applet is an extension of the *java.applet.Applet class*. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following −

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may −

- Request information about the author, version, and copyright of the applet
- Request a description of the parameters the applet recognizes
- Initialize the applet
- Destroy the applet
- Start the applet's execution
- Stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

The "Hello, World" applet is complete as it stands. The only method overridden is the paint method.

Invoking an Applet

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.

The <applet> tag is the basis for embedding an applet in an HTML file. Following is an example that invokes the "Hello, World" applet −

```
<html>
```

```
   <title>The Hello, World Applet</title>
   <hr>
   <applet code = "HelloWorldApplet.class" width = "320" height = "120">
      If your browser was Java-enabled, a "Hello, World"
      message would appear here.
   </applet>
   <hr>
</html>
```

The code attribute of the <applet> tag is required. It specifies the Applet class to run. Width and height are also required to specify the initial size of the panel in which an applet runs. The applet directive must be closed with an </applet> tag.

If an applet takes parameters, values may be passed for the parameters by adding <param> tags between <applet> and </applet>. The browser ignores text and other tags between the applet tags.

Non-Java-enabled browsers do not process <applet> and </applet>. Therefore, anything that appears between the tags, not related to the applet, is visible in non-Java-enabled browsers.

The viewer or browser looks for the compiled Java code at the location of the document. To specify otherwise, use the codebase attribute of the <applet> tag as shown −

```
<applet codebase = "https://amrood.com/applets" code = "HelloWorldApplet.class"
   width = "320" height = "120">
```

If an applet resides in a package other than the default, the holding package must be specified in the code attribute using the period character (.) to separate package/class components. For example −

```
<applet = "mypackage.subpackage.TestApplet.class"
   width = "320" height = "120">
```

Getting Applet Parameters

The following example demonstrates how to make an applet respond to setup parameters specified in the document. This applet displays a checkerboard pattern of black and a second color.

The second color and the size of each square may be specified as parameters to the applet within the document.

CheckerApplet gets its parameters in the init() method. It may also get its parameters in the paint() method. However, getting the values and saving the settings once at the start of the applet, instead of at every refresh, is convenient and efficient.

The applet viewer or browser calls the init() method of each applet it runs. The viewer calls init() once, immediately after loading the applet. (Applet.init() is implemented to do nothing.) Override the default implementation to insert custom initialization code.

The Applet.getParameter() method fetches a parameter given the parameter's name (the value of a parameter is always a string). If the value is numeric or other non-character data, the string must be parsed.

The following is a skeleton of CheckerApplet.java −

```
import java.applet.*;
import java.awt.*;

public class CheckerApplet extends Applet {
   int squareSize = 50;   // initialized to default size
   public void init() {}
   private void parseSquareSize (String param) {}
   private Color parseColor (String param) {}
   public void paint (Graphics g) {}
}
```

Here are CheckerApplet's init() and private parseSquareSize() methods −

```
public void init () {
   String squareSizeParam = getParameter ("squareSize");
   parseSquareSize (squareSizeParam);

   String colorParam = getParameter ("color");
   Color fg = parseColor (colorParam);

   setBackground (Color.black);
   setForeground (fg);
}

private void parseSquareSize (String param) {
   if (param == null) return;
   try {
```

```
    squareSize = Integer.parseInt (param);
  } catch (Exception e) {
    // Let default value remain
  }
}
```

The applet calls parseSquareSize() to parse the squareSize parameter. parseSquareSize() calls the library method Integer.parseInt(), which parses a string and returns an integer. Integer.parseInt() throws an exception whenever its argument is invalid.

Therefore, parseSquareSize() catches exceptions, rather than allowing the applet to fail on bad input.

The applet calls parseColor() to parse the color parameter into a Color value. parseColor() does a series of string comparisons to match the parameter value to the name of a predefined color. You need to implement these methods to make this applet work.

Specifying Applet Parameters

The following is an example of an HTML file with a CheckerApplet embedded in it. The HTML file specifies both parameters to the applet by means of the <param> tag.

```
<html>
  <title>Checkerboard Applet</title>
  <hr>
  <applet code = "CheckerApplet.class" width = "480" height = "320">
    <param name = "color" value = "blue">
    <param name = "squaresize" value = "30">
  </applet>
  <hr>
</html>
```

Note − Parameter names are not case sensitive.

Application Conversion to Applets

It is easy to convert a graphical Java application (that is, an application that uses the AWT and that you can start with the Java program launcher) into an applet that you can embed in a web page.

Following are the specific steps for converting an application to an applet.

- Make an HTML page with the appropriate tag to load the applet code.
- Supply a subclass of the JApplet class. Make this class public. Otherwise, the applet cannot be loaded.
- Eliminate the main method in the application. Do not construct a frame window for the application. Your application will be displayed inside the browser.
- Move any initialization code from the frame window constructor to the init method of the applet. You don't need to explicitly construct the applet object. The browser instantiates it for you and calls the init method.
- Remove the call to setSize; for applets, sizing is done with the width and height parameters in the HTML file.
- Remove the call to setDefaultCloseOperation. An applet cannot be closed; it terminates when the browser exits.
- If the application calls setTitle, eliminate the call to the method. Applets cannot have title bars. (You can, of course, title the web page itself, using the HTML title tag.)
- Don't call setVisible(true). The applet is displayed automatically.

Event Handling

Applets inherit a group of event-handling methods from the Container class. The Container class defines several methods, such as processKeyEvent and processMouseEvent, for handling particular types of events, and then one catch-all method called processEvent.

In order to react to an event, an applet must override the appropriate event-specific method.

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.applet.Applet;
import java.awt.Graphics;

public class ExampleEventHandling extends Applet implements MouseListener {
   StringBuffer strBuffer;
```

```
public void init() {
  addMouseListener(this);
  strBuffer = new StringBuffer();
  addItem("initializing the apple ");
}

public void start() {
  addItem("starting the applet ");
}

public void stop() {
  addItem("stopping the applet ");
}

public void destroy() {
  addItem("unloading the applet");
}

void addItem(String word) {
  System.out.println(word);
  strBuffer.append(word);
  repaint();
}

public void paint(Graphics g) {
  // Draw a Rectangle around the applet's display area.
  g.drawRect(0, 0,
  getWidth() - 1,
  getHeight() - 1);

  // display the string inside the rectangle.
  g.drawString(strBuffer.toString(), 10, 20);
}


public void mouseEntered(MouseEvent event) {
}
public void mouseExited(MouseEvent event) {
}
public void mousePressed(MouseEvent event) {
```

```
   }
   public void mouseReleased(MouseEvent event) {
   }
   public void mouseClicked(MouseEvent event) {
      addItem("mouse clicked! ");
   }
}
```

Now, let us call this applet as follows −

```
<html>
   <title>Event Handling</title>
   <hr>
   <applet code = "ExampleEventHandling.class"
      width = "300" height = "300">
   </applet>
   <hr>
</html>
```

Initially, the applet will display "initializing the applet. Starting the applet." Then once you click inside the rectangle, "mouse clicked" will be displayed as well.

Displaying Images

An applet can display images of the format GIF, JPEG, BMP, and others. To display an image within the applet, you use the drawImage() method found in the java.awt.Graphics class.

Following is an example illustrating all the steps to show images −

```
import java.applet.*;
import java.awt.*;
import java.net.*;

public class ImageDemo extends Applet {
   private Image image;
   private AppletContext context;

   public void init() {
      context = this.getAppletContext();
      String imageURL = this.getParameter("image");
      if(imageURL == null) {
```

```
      imageURL = "java.jpg";
    }
    try {
      URL url = new URL(this.getDocumentBase(), imageURL);
      image = context.getImage(url);
    } catch (MalformedURLException e) {
      e.printStackTrace();
      // Display in browser status bar
      context.showStatus("Could not load image!");
    }
  }

  public void paint(Graphics g) {
    context.showStatus("Displaying image");
    g.drawImage(image, 0, 0, 200, 84, null);
    g.drawString("www.javalicense.com", 35, 100);
  }
}
```

Now, let us call this applet as follows −

```
<html>
  <title>The ImageDemo applet</title>
  <hr>
  <applet code = "ImageDemo.class" width = "300" height = "200">
    <param name = "image" value = "java.jpg">
  </applet>
  <hr>
</html>
```

Playing Audio

An applet can play an audio file represented by the AudioClip interface in the java.applet package. The AudioClip interface has three methods, including −

- **public void play()** − Plays the audio clip one time, from the beginning.
- **public void loop()** − Causes the audio clip to replay continually.
- **public void stop()** − Stops playing the audio clip.

To obtain an AudioClip object, you must invoke the getAudioClip() method of the Applet class. The getAudioClip() method returns immediately, whether or not the

URL resolves to an actual audio file. The audio file is not downloaded until an attempt is made to play the audio clip.

Following is an example illustrating all the steps to play an audio −

```
import java.applet.*;
import java.awt.*;
import java.net.*;

public class AudioDemo extends Applet {
   private AudioClip clip;
   private AppletContext context;

   public void init() {
      context = this.getAppletContext();
      String audioURL = this.getParameter("audio");
      if(audioURL == null) {
         audioURL = "default.au";
      }
      try {
         URL url = new URL(this.getDocumentBase(), audioURL);
         clip = context.getAudioClip(url);
      } catch (MalformedURLException e) {
         e.printStackTrace();
         context.showStatus("Could not load audio file!");
      }
   }

   public void start() {
   if(clip != null) {
   clip.loop();
      }
   }

   public void stop() {
   if(clip != null) {
   clip.stop();
      }
   }
}
```

Now, let us call this applet as follows −

```
<html>
   <title>The ImageDemo applet</title>
   <hr>
   <applet code = "ImageDemo.class" width = "0" height = "0">
      <param name = "audio" value = "test.wav">
   </applet>
   <hr>
</html>
```

Java Servlet

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Sample Code

Following is the sample source code structure of a servlet example to show Hello World −

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

   private String message;

   public void init() throws ServletException {
      // Do required initialization
      message = "Hello World";
   }
```

```
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    // Actual logic goes here.
    PrintWriter out = response.getWriter();
    out.println("<h1>" + message + "</h1>");
  }

  public void destroy() {
    // do nothing.
  }
}
```

Compiling a Servlet

Let us create a file with name HelloWorld.java with the code shown above. Place this file at C:\ServletDevel (in Windows) or at /usr/ServletDevel (in Unix). This path location must be added to CLASSPATH before proceeding further.

Assuming your environment is setup properly, go in **ServletDevel** directory and compile HelloWorld.java as follows −

```
$ javac HelloWorld.java
```

If the servlet depends on any other libraries, you have to include those JAR files on your CLASSPATH as well. I have included only servlet-api.jar JAR file because I'm not using any other library in Hello World program.

This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK). For this command to work properly, you have to include the location of the Java SDK that you are using in the PATH environment variable.

If everything goes fine, above compilation would produce **HelloWorld.class** file in the same directory. Next section would explain how a compiled servlet would be deployed in production.

Servlet Deployment

By default, a servlet application is located at the path <Tomcat-installationdirectory>/webapps/ROOT and the class file would reside in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>HelloWorld</servlet-name>
  <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installationdirectory>\bin\startup.bat (on Windows) or <Tomcat-installationdirectory>/bin/startup.sh (on Linux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in the browser's address box. If everything goes fine, you would get the following result

Introduction to ASP.NET Technology

ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC, as well as mobile devices.

ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.

ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

The ASP.NET application codes can be written in any of the following languages:

- C#
- Visual Basic.Net
- Jscript
- J#

ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.

ASP.NET Web Forms Model

ASP.NET web forms extend the event-driven model of interaction to the web applications. The browser submits a web form to the web server and the server returns a full markup page or HTML page in response.

All client side user activities are forwarded to the server for stateful processing. The server processes the output of the client actions and triggers the reactions.

Now, HTTP is a stateless protocol. ASP.NET framework helps in storing the information regarding the state of the application, which consists of:

- Page state
- Session state

The page state is the state of the client, i.e., the content of various input fields in the web form. The session state is the collective information obtained from

various pages the user visited and worked with, i.e., the overall session state. To clear the concept, let us take an example of a shopping cart.

User adds items to a shopping cart. Items are selected from a page, say the items page, and the total collected items and price are shown on a different page, say the cart page. Only HTTP cannot keep track of all the information coming from various pages. ASP.NET session state and server side infrastructure keeps track of the information collected globally over a session.

The ASP.NET runtime carries the page state to and from the server across page requests while generating ASP.NET runtime codes, and incorporates the state of the server side components in hidden fields.

This way, the server becomes aware of the overall application state and operates in a two-tiered connected way.

The ASP.NET Component Model

The ASP.NET component model provides various building blocks of ASP.NET pages. Basically it is an object model, which describes:

- Server side counterparts of almost all HTML elements or tags, such as <form> and <input>.
- Server controls, which help in developing complex user-interface. For example, the Calendar control or the Gridview control.

ASP.NET is a technology, which works on the .Net framework that contains all web-related functionalities. The .Net framework is made of an object-oriented hierarchy. An ASP.NET web application is made of pages. When a user requests an ASP.NET page, the IIS delegates the processing of the page to the ASP.NET runtime system.

The ASP.NET runtime transforms the .aspx page into an instance of a class, which inherits from the base class page of the .Net framework. Therefore, each ASP.NET page is an object and all its components i.e., the server-side controls are also objects.

Components of .Net Framework 3.5

Before going to the next session on Visual Studio.Net, let us go through at the various components of the .Net framework 3.5. The following table describes the components of the .Net framework 3.5 and the job they perform:

Components and their Description
(1) Common Language Runtime or CLR It performs memory management, exception handling, debugging, security checking, thread execution, code execution, code safety, verification, and compilation. The code that is directly managed by the CLR is called the managed code. When the managed code is compiled, the compiler converts the source code into a CPU independent intermediate language (IL) code. A Just In Time(JIT) compiler compiles the IL code into native code, which is CPU specific.
(2) .Net Framework Class Library It contains a huge library of reusable types. classes, interfaces, structures, and enumerated values, which are collectively called types.
(3) Common Language Specification It contains the specifications for the .Net supported languages and implementation of language integration.
(4) Common Type System It provides guidelines for declaring, using, and managing types at runtime, and cross-language communication.
(5) Metadata and Assemblies Metadata is the binary information describing the program, which is either stored in a portable executable file (PE) or in the memory. Assembly is a logical unit consisting of the assembly manifest, type metadata, IL code, and a set of resources like image files.
(6) Windows Forms Windows Forms contain the graphical representation of any window displayed in the application.

(7) ASP.NET and ASP.NET AJAX

ASP.NET is the web development model and AJAX is an extension of ASP.NET for developing and implementing AJAX functionality. ASP.NET AJAX contains the components that allow the developer to update data on a website without a complete reload of the page.

(8) ADO.NET

It is the technology used for working with data and databases. It provides access to data sources like SQL server, OLE DB, XML etc. The ADO.NET allows connection to data sources for retrieving, manipulating, and updating data.

(9) Windows Workflow Foundation (WF)

It helps in building workflow-based applications in Windows. It contains activities, workflow runtime, workflow designer, and a rules engine.

(10) Windows Presentation Foundation

It provides a separation between the user interface and the business logic. It helps in developing visually stunning interfaces using documents, media, two and three dimensional graphics, animations, and more.

(11) Windows Communication Foundation (WCF)

It is the technology used for building and executing connected systems.

(12) Windows CardSpace

It provides safety for accessing resources and sharing personal information on the internet.

(13) LINQ

It imparts data querying capabilities to .Net languages using a syntax

which is similar to the tradition query language SQL.

.NET vs Competing Technologies

Microsoft .NET and Java are two leading technologies for building softwares, websites and web apps. With their growing popularity, most of the businesses face the challenge to choose from either of them as a primary development tool for creating intuitive applications. Both these technologies enable the creation of large-scale business applications and have evolved over the years to support and enhance desktop & server-side application development.

Between the two, you must first learn about the applications created using either of the technologies. Through this comparison blog, we aim to highlight the difference between Java and .NET. Our objective is to help businesses understand how either of the environments can fit in their operational requirements.
A .NET or Microsoft technology-based solution might be the correct choice for an Enterprise-grade application which requires strict security and high level of data integrity, whereas a Java-based solution would be suitable when the primary requirement is cross-platform operability in-line with its motto "write once, deploy anywhere." Learn more about the new features of ASP.NET Core that enable development of modern web & cloud applications.

.NET vs Java – The Differences

Java is object-oriented programming (OOPs) language and .NET is a framework with C# as its programming language. Both Java and .NET are based on the object-oriented concept and are useful for developing enterprise solutions.

Feature	Microsoft .NET	Java
Programming Languages	C#, VB.NET, C++, .NET, PHP, Ruby, Python & more	Java, Clojure, Groovy, Scala, PHP, Ruby, Python, JavaScript & more
Runtime	CLR	JVM
Supported IDE	Microsoft Visual Studio,	Eclipse, IntelliJ Idea, Oracle

	Rider, MonoDevelop	NetBeans, and Oracle JDeveloper
Server Components	.NET COM, OLE Automation	EJBs, JCA, JMX
GUI Frameworks	WPF, WinForms, UWP	JavaFX, Swing GUI Java, AWT, SWT & more
Web Services Support	Built-in	Add-on
Unit Testing	Microsoft Unit Testing Framework, NUnit	JUnit
Web Application Framework	ASP.NET Core, Spring .NET	Spring, Apache Wicket, JSF, Struts & more
Web Server Scripting	ASP.NET	JSF
Data Access	ADO.NET, oLeDB, Dapper & more	JDBC
HTTP Engine	IIS	Application Servers from Multiple Vendors, Glassfish, Tomcat & More
Remoting	SOAP, OpenAPI, DCOM & More	RMI, Rest API, GraphQL & More

Request Call Back

Both are intended to simplify the development of apps by providing a system of modular, standardized components and services. The following head-to-head Java vs .NET comparison can help businesses make a better choice.

.NET Framework vs. Java

- **Platform**
  Multi-platform portability is ensured by compiling source code into an intermediate language executable by all Java Virtual Machines (JVM). The JVM translates the code into bytecode making it compatible with the machine code according to the operating system on which it is installed.
  The .NET framework is secure than most of the open-source platforms. It comes with the Common Language Runtime (CLR) framework, which supports the use of components, developed in multiple languages. Further, Microsoft has even developed a CLR engine that converts the program code into the Microsoft Intermediate Language (MSIL) and finally "just in time," translates it into native machine code.
- **Language Support**
  The peculiarity of Java lies in the sharing of a single language across different platforms. However, the programs written with it work independently across different OS types. Java supports programming and scripting languages like Python, Ruby, Groovy, Scala, and Kotlin.
  The DotNet framework supports languages especially for backend development and web services, like C# and C++. It currently supports about 20 languages. It allows you to program in any language you choose (including, Vb.NET, C # .NET, Perl and many others). However, it generates a specific code for the Windows platform only.
  Therefore, while .NET supports a multi-programming environment; Java is focused on a single programming language that supports multiple environments. Learn more about the top Java benefits for SMBs.
- **IDE**
  Java IDE comprises of a code editor, compiler, and debugger. Eclipse, IntelliJ Idea, Oracle NetBeans, and Oracle JDeveloper are the main IDEs designed to make writing and testing of the code easier. These IDEs come with inbuilt plugins and auto-fill options, which boosts Java's flexibility and provides scope for innovation.
  The .NET platform is integrated with Visual Studio, which allows editing, compiling, and run-time customization of the behavior of APIs using

standard library macros. Further, developers do not need to evaluate the IDEs and other tools in advance.

- **Third-Party Integrators**
  Java facilitates the integration of third-party tools and offers developers the freedom to choose their choice of OS during development.
  .NET offers integrated services like SharePoint and Microsoft Exchange. It is rich in functions designed for creating applications on the Windows platform. Interoperability is unique to .NET, which allows its applications to run seamlessly on other platforms too.
- **Performance & Compatibility**
  Java does not require conversion to machine language until the code gets executed. While .NET is compiled and then run on the system where they are deployed. That is why C# works better in a runtime environment.
  Versions of Java older than Java7 does not support the simplified data structure, switch case.
  .NET supports switch case for string variable in C# and native generic data. Query wise, Linq (Language Integrated Query) is not supported in Java unlike in .NET, but the latter allows questions on stored procedures.
- **Security**
  For open source platforms like Java, security always remains a concern due to a lack of professional support. Since .NET is of a proprietary platform, Microsoft takes care of the security aspect. It provides round-the-clock support for its enterprise clients.

JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages.

JavaScript code is not compiled but translated by the translator. This translator is embedded into the browser and is responsible for translating javascript code.

Key Points

- It is Lightweight, interpreted programming language.
- It is designed for creating network-centric applications.
- It is complementary to and integrated with Java.
- It is complementary to and integrated with HTML
- It is an open and cross-platform

JavaScript Statements

JavaScript statements are the commands to tell the browser to what action to perform. Statements are separated by semicolon (;).

JavaScript statement constitutes the JavaScript code which is translated by the browser line by line.

Example of JavaScript statement:

```
document.getElementById("demo").innerHTML = "Welcome";
```

Following table shows the various JavaScript Statements −

Sr.No.	Statement	Description
1.	switch case	A block of statements in which execution of code depends upon different cases. 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.
2.	If else	The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.
3.	While	The purpose of a while loop is to execute a statement or code block repeatedly as long as expression is true. Once expression becomes false, the loop will be exited.
4.	do while	Block of statements that are executed at least once and continues to be executed while condition is true.
5.	for	Same as while but initialization, condition and increment/decrement is done in the same line.
6.	for in	This loop is used to loop through an object's properties.
7.	continue	The continue statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.
8.	break	The break statement is used to exit a loop early, breaking out of the enclosing curly braces.
9.	function	A function is a group of reusable code which can be called anywhere in your programme. The keyword function is used to declare a function.
10.	return	Return statement is used to return a value from a function.
11.	var	Used to declare a variable.
12.	Try	A block of statements on which error handling is implemented.
13.	Catch	A block of statements that are executed when an error occur.
14.	throw	Used to throw an error.

JavaScript Comments

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

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

      // this is a comment. It is similar to comments in C++

      /*
         * This is a multiline comment in JavaScript
         * It is very similar to comments in C Programming
      */
   //-->
<script>
```

JavaScript variable

Variables are referred as named containers for storing information. We can place data into these containers and then refer to the data simply by naming the container.

Rules to declare variable in JavaScript

Here are the important rules that must be followed while declaring a variable in JavaScript.

- In JavaScript variable names are case sensitive i.e. a is different from A.
- Variable name can only be started with a underscore (_) or a letter (from a to z or A to Z), or dollar ($) sign.
- Numbers (0 to 9) can only be used after a letter.
- No other special character is allowed in variable name.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows −

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

Variables can be initialized at time of declaration or after declaration as follows −

```
<script type="text/javascript">
   <!--
      var name = "Ali";
      var money;
      money = 2000.50;
   //-->
</script>
```

Javascript Data Type

There are two kinds of data types as mentioned below −

- Primitive Data Type
- Non Primitive Data Type

The following table describes **Primitive Data Types** available in javaScript

Sr.No.	Datatype Description
1.	**String** Can contain groups of character as single value. It is represented in double quotes.E.g. var x= “tutorial”.
2.	**Numbers** Contains the numbers with or without decimal. E.g. var x=44, y=44.56;
3.	**Booleans** Contain only two values either true or false. E.g. var x=true, y= false.
4.	**Undefined** Variable with no value is called Undefined. E.g. var x;
5.	**Null** If we assign null to a variable, it becomes empty. E.g. var x=null;

The following table describes **Non-Primitive Data Types** in javaScript

Sr.No.	Datatype Description
1.	**Array** Can contain groups of values of same type. E.g. var x={1,2,3,55};
2.	**Objects** Objects are stored in property and value pair. E.g. var rectangle = { length: 5, breadth: 3};

JavaScript Functions

Function is a group of reusable statements (Code) that can be called any where in a program. In javascript function keyword is used to declare or define a function.

Key Points

- To define a function use function keyword followed by functionname, followed by parentheses ().
- In parenthesis, we define parameters or attributes.
- The group of reusabe statements (code) is enclosed in curly braces {}. This code is executed whenever function is called.

Syntax

```
function functionname (p1, p2) {
   function coding…
}
```

JavaScript Operators

Operators are used to perform operation on one, two or more operands. Operator is represented by a symbol such as +, =, *, % etc. Following are the operators supported by javascript −

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

Arithmatic Operators

Following table shows all the arithmetic operators supported by javascript −

Operator	Description	Example
+	Add two operands.	10 + 10 will give 20
-	Subtract second operand from the first.	10 – 10 will give 0
*	Multiply two operands.	10 * 30 will give 300
/	Divide numerator by denominator	10/10 will give 1
%	It is called modulus operator and gives remainder of the division.	10 % 10 will give 0
++	Increment operator, increases integer value by one	10 ++ will give 11
--	Decrement operator, decreases integer value by one	10 – will give 9

Comparison Operators

Following table shows all the comparison operators supported by javascript −

Operator	Description	Example
==	Checks if values of two operands are equal or not, If yes then condition becomes true.	10 == 10 will give true
!=	**Not Equal to operator** Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	10 !=10 will give false
>	**Greater Than operator** Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	20 > 10 will give true
<	**Less than operator** Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	10 < 20 will give true
>=	**Greater than or equal to operator** Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	10 >=20 will give false
<=	**Less than or equal to operator** Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	10 <=20 will give true.

Logical Operators

Following table shows all the logical operators supported by javascript −

Operator	Description	Example
&&	Logical **AND** operator returns true if both operands are non zero.	10 && 10 will give true.
||	Logical **OR** operator returns true If any of the operand is non zero	10 || 0 will give true.
!	Logical **NOT** operator complements the logical state of its operand.	! (10 && 10) will give false.

Assignment Operators

Following table shows all the assignment operators supported by javascript −

Operator	Description	Example
=	**Simple Assignment operator** Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	**Add AND assignment operator** It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	**Subtract AND assignment operator** It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	**Multiply AND assignment operator** It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	**Divide AND assignment operator** It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	**Modulus AND assignment operator** Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Conditional Operator

It is also called ternary operator, since it has three operands.

Operator	Description	Example
?:	Conditional Expression	If Condition is true? Then value X : Otherwise value Y

Control Structure

Control structure actually controls the flow of execution of a program. Following are the several control structure supported by javascript.

- if … else
- switch case
- do while loop
- while loop
- for loop

If … else

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

Syntax

```
if (expression){
```

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

Example

```
<script type="text/javascript">
   <!--
      var age = 20;
      if( age > 18 ){
         document.write("<b>Qualifies for driving</b>");
      }
   //-->
</script>
```

Switch case

The basic syntax of the 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.

Syntax

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

Example

```
<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>
```

Do while Loop

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

Syntax

```
do{
   Statement(s) to be executed;
} while (expression);
```

Example

```
<script type="text/javascript">
   <!--
      var count = 0;
      document.write("Starting Loop" + "<br/>");
      do{
         document.write("Current Count : " + count + "<br/>");
         count++;
      }while (count < 0);
      document.write("Loop stopped!");
   //-->
</script>
```

This will produce following result −

```
Starting Loop
Current Count : 0
Loop stopped!
```

While Loop

The purpose of a while loop is to execute a statement or code block repeatedly as long as expression is true. Once expression becomes false, the loop will be exited.

Syntax

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

Example

```
<script type="text/javascript">
   <!--
      var count = 0;
      document.write("Starting Loop" + "<br/>");
      while (count < 10){
         document.write("Current Count : " + count + "<br/>");
         count++;
      }
      document.write("Loop stopped!");
   //-->
</script>
```

This will produce following result −

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
```

```
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

For Loop

The for loop is the most compact form of looping and includes the following three important parts −

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.

Syntax

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

Example

```
<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>
```

This will produce following result which is similar to while loop −

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

Creating Sample Program

Following is the sample program that shows time, when we click in button.

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

```
      <button onclick="this.innerHTML=Date()">The time is?</button>
      <p>Click to display the date.</p>
      <button onclick="displayDate()">The time is?</button>
      <script>
         function displayDate() {
            document.getElementById("demo").innerHTML = Date();
         }</script>

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

Output

https://www.tutorialspoint.com/internet_technologies/
https://www.javatpoint.com/javascript-tutorial