

Unit – I: Web Environment

Internetworking concepts – Devices – Repeaters – Bridges – Routers – Gateways – Internet topology Internal Architecture of an ISP – IP Address – Basics of TCP – Features of TECP – UDP – DNS – Email – FTP – HTTP – TELNET - Web Server and its deployment- N-Tier Arch.- Services of Web Server – Mail server- News server- Proxy server- Multimedia server-

Unit – II: HTML and XML

Formatting- tags- links- list- tables- frames- forms- comments in HTML.

XML: Introduction- displaying an XML Document- Data interchange with an XML Document- Document type definition- Parsers using XML- Client-side usage- Server- side Usage.

Unit – III: Java Script

Introduction- Documents- forms- Statements- Functions- Objects in Java scripts- events and event handling- arrays- FORMS- Buttons- Checkboxes- Text fields and text areas.

Unit – IV: JSP

JSP: JSP overview- JSP language basics- JSP translation and compilation directives- Standard java objects from JSP- JSP configuration and deployment- actions and tags of JSP; Java servlets – Arch- servlet interface- applications of servlets.

Unit – V: VB Script

VBScript in the body of the HTML – Variables - Assignments and expression Procedures and functions-Decisional (conditional/alternative) statements List of VBScript intrinsic functions

Text Books:

1. Phil Hanna, “Instant Java Servlets”, Tata McGraw Hill 2000
2. William B.Brogden Bill Brogden- Chris Minnick,”Java Developer's Guide to E- Commerce with XML and JSP”, Sybex book, 2001
3. Stephen Walther and others, “Active Server Pages Unleashed”, Wrox press Ltd ,1998.

Reference Books:

1. John Wiley , “COM+ & XML: ASP.Net on the Edge” 2001
2. Burdman- “Collaborative Web Development”- Addison Wesley,.1999
3. Sharma & Sharma- “Developing E-Commerce Sites”- Addison Wesley,. 2000
4. Ivan Bayross- “Web Technologies Part II”- BPB Publications. McGraw Hill 2004

5. Shishir Gundavarma- “CGI Programming on the World Wide Web”- O'Reilly & Associate,. 1996
6. DON Box- “Essential COM”- Addison Wesley,1998
7. Greg Buczek- “ASP Developer's Guide”, Tata McGraw-Hill, 2000

Unit-1

Web Environment

Network Devices (Hub, Repeater, Bridge, Switch, Router, Gateways and Brouter)

1. Repeater – A repeater operates at the physical layer. Its job is to regenerate the signal over the same network before the signal becomes too weak or corrupted so as to extend the length to which the signal can be transmitted over the same network.

An important point to be noted about repeaters is that they do not amplify the signal. When the signal becomes weak, they copy the signal bit by bit and regenerate it at the original strength. It is a 2 port device.

2. Hub – A hub is basically a multiport repeater. A hub connects multiple wires coming from different branches, for example, the connector in star topology which connects different stations. Hubs cannot filter data, so data packets are sent to all connected devices. In other words, collision domain of all hosts connected through Hub remains one. Also, they do not have intelligence to find out best path for data packets which leads to inefficiencies and wastage.

Types of Hub

- **Active Hub:-** These are the hubs which have their own power supply and can clean, boost and relay the signal along with the network. It serves both as a repeater as well as wiring centre. These are used to extend the maximum distance between nodes.
- **Passive Hub :-** These are the hubs which collect wiring from nodes and power supply from active hub. These hubs relay signals onto the network without cleaning and boosting them and can't be used to extend the distance between nodes.

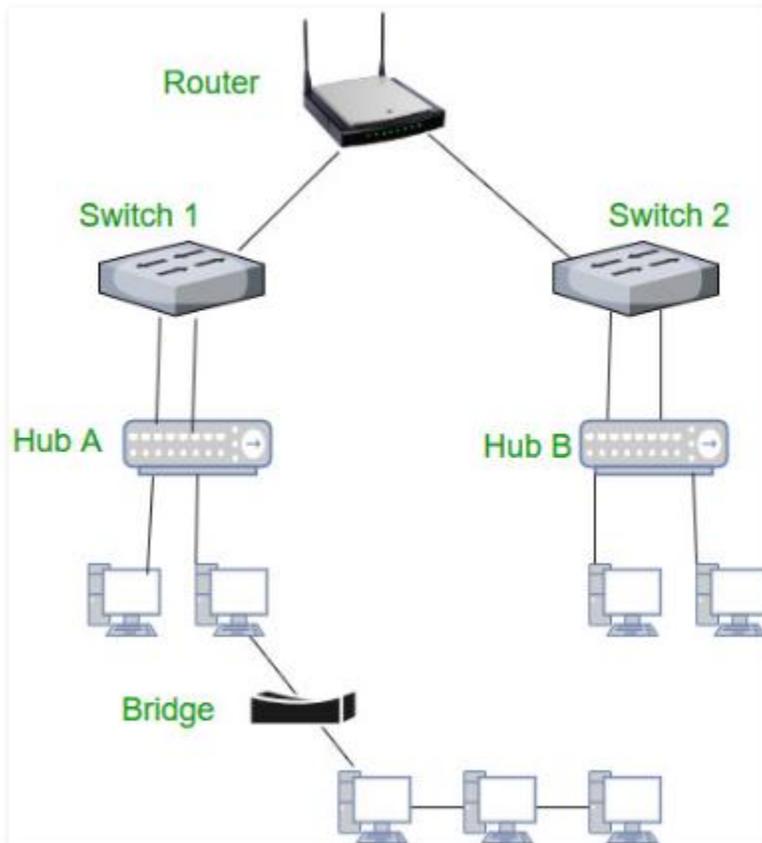
3. Bridge – A bridge operates at data link layer. A bridge is a repeater, with add on the functionality of filtering content by reading the MAC addresses of source and destination. It is also used for interconnecting two LANs working on the same protocol. It has a single input and single output port, thus making it a 2 port device.

Types of Bridges

- **Transparent Bridges:-** These are the bridge in which the stations are completely unaware of the bridge's existence i.e. whether or not a bridge is added or deleted from the network, reconfiguration of the stations is unnecessary. These bridges make use of two processes i.e. bridge forwarding and bridge learning.
- **Source Routing Bridges:-** In these bridges, routing operation is performed by source station and the frame specifies which route to follow. The host can discover frame by sending a special frame called discovery frame, which spreads through the entire network using all possible paths to destination.

4. Switch – A switch is a multiport bridge with a buffer and a design that can boost its efficiency (a large number of ports imply less traffic) and performance. A switch is a data link layer device. The switch can perform error checking before forwarding data, that makes it very efficient as it does not forward packets that have errors and forward good packets selectively to correct port only. In other words, switch divides collision domain of hosts, but broadcast domain remains same

5. Routers – A router is a device like a switch that routes data packets based on their IP addresses. Router is mainly a Network Layer device. Routers normally connect LANs and WANs together and have a dynamically updating routing table based on which they make decisions on routing the data packets. Router divide broadcast domains of hosts connected through it.



6. Gateway – A gateway, as the name suggests, is a passage to connect two networks together that may work upon different networking models. They basically work as the messenger agents that take data from one system, interpret it, and transfer it to another system. Gateways are also called protocol converters and can operate at any network layer. Gateways are generally more complex than switch or router.

7. Brouter – It is also known as bridging router is a device which combines features of both bridge and router. It can work either at data link layer or at network layer. Working as router, it is capable of routing packets across networks and working as bridge, it is capable of filtering local area network traffic.

Internet - Topologies

The Internet is a network of networks. These networks are connected using routers. There are a number of different layouts, or topologies, used in the networks that are connected to the Internet.

The main types are:

- bus
- ring
- star.

Bus topology

All the computers, or nodes, or ports, or hosts, are linked together using a common bus, or set of wires.

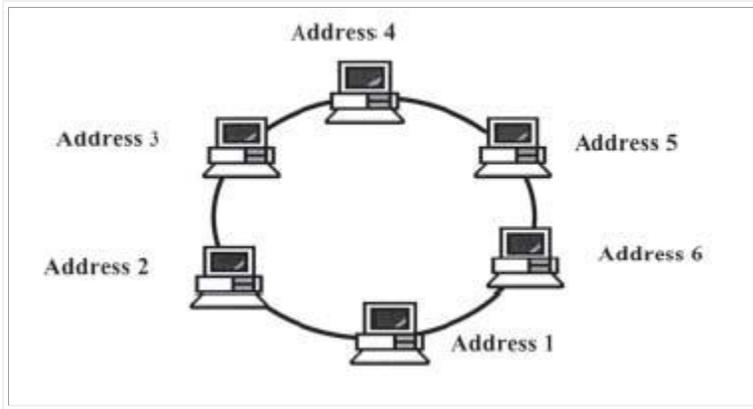


Each host has a unique hardware address. **Packets** are sent out onto the **bus**. The data packet contains the address of the sender and receiver. The receiver will only access the data once the address has been read.

The terminator boxes are needed to stop unread packets from being ‘bounced back’ into the bus and causing data errors.

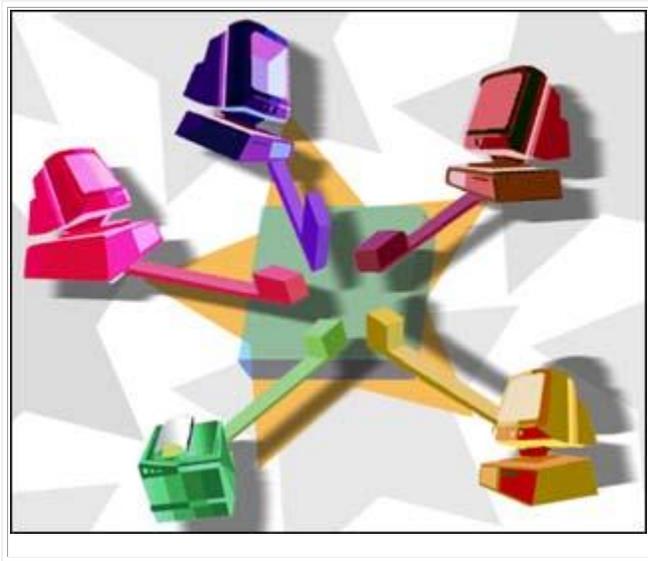
Ring topology

All of the computers/nodes/ports are connected to form a ring.



If the computer with **address 1** wants to send a message to the computer with **address 5** then the message is routed to machine 2, then machine 3 and machine 4 and eventually to machine 5. On reading the **IP**, each of the intermediate computers will realise that the message is not to be accessed and it is passed on around the ring.

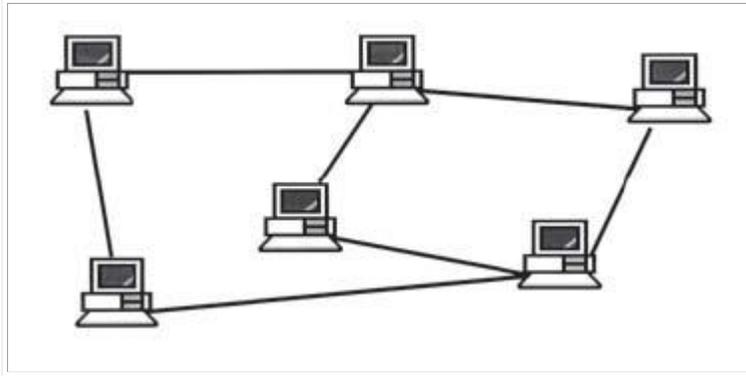
Star topology



A star topology requires the use of a **hub** or a **switch**, which is located at the centre of the network. The hub or switch is situated at the centre of the network and computers communicate by passing information directly onto the hub.

Mesh topology

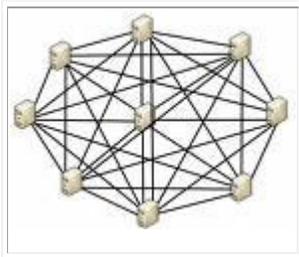
The Internet uses a mesh structure. A mesh topology requires that each machine be linked to at least two other machines.



If one of the computers or links is missing, the message can still be transmitted.

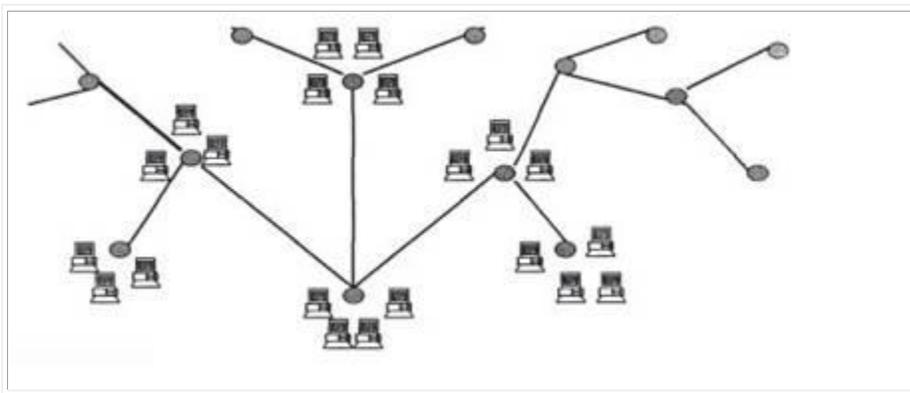
Fully meshed

A fully meshed topology requires that each computer be linked to **every other** computer.



Tree topology

The Internet uses tree topology. A tree topology consists of a number of star topologies linked together.



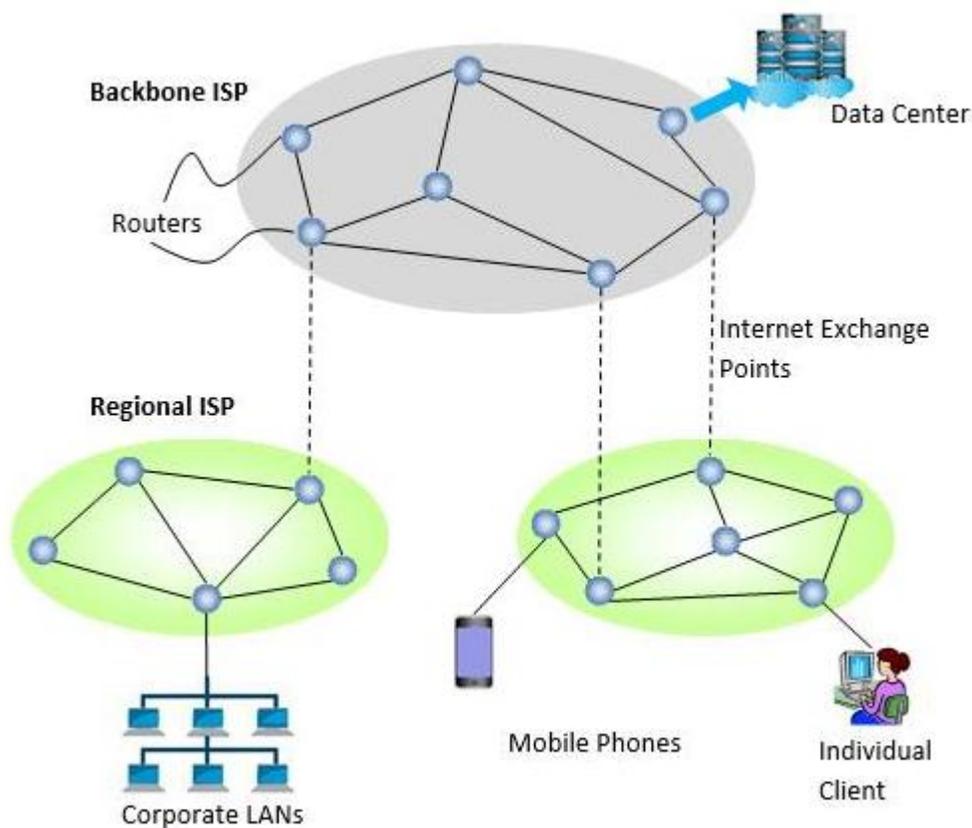
Internal Architecture of ISP

The architecture of the Internet is ever-changing due to continuous changes in the technologies as well as the nature of the service provided. The heterogeneity and vastness of the Internet make it difficult to describe every aspect of its architecture.

The overall architecture can be described in three levels –

1. Backbone ISP (Internet Service Provider)
2. Regional ISPs
3. Clients

The following diagram shows the three levels –



Backbone ISP (Internet Service Provider) – Backbone ISPs are large international backbone networks. They are equipped with thousands of routers and store enormous amounts of information in data centers, connected through high bandwidth fiber optic links. Everyone needs to connect with a backbone ISP to access the entire Internet.

There are different ways through which a client can connect to the ISP. A commonly used way is DSL (Digital Subscriber Line) which reuses the telephone connection of the user for transmission of digital data. The user uses a dial-up connection instead of the telephone call. Connectivity is also done by sending signals over cable TV system that reuses unused cable TV channels for data transmission. For high-speed Internet access, the connectivity can be done through FTTH (Fiber to the Home), that uses optical fibers for transmitting data. Nowadays, most Internet

access is done through the wireless connection to mobile phones from fixed subscribers, who transmit data within their coverage area.

Internet Address Architecture

The structure of network-layer addresses used in the Internet, the IP addresses. [p31-32]

- Every device connected to the Internet has at least one IP address.
- When devices are attached to the global Internet, they are assigned addresses that must be coordinated so as to not duplicate other addresses in use on the network.

Expressing IP Addresses

In IPv4, the dotted-quad notation for IPv4 addresses consists of four decimal numbers separated by periods. For example, 165.195.130.107. Each such number is a nonnegative integer in the range [0, 255] and represents one-quarter of the entire IP address. It is simply a way of writing the whole IPv4 address (a 32-bit nonnegative integer used throughout the Internet system) using convenient decimal numbers. [p32]

In IPv6, addresses are 128 bits in length, four times larger than IPv4 addresses. The conventional notation for IPv6 addresses is a series of four hexadecimal ("hex" or base-16) numbers called *blocks* or *fields* separated by colons. For example, an IPv6 address containing eight blocks would be written as 5f05:2000:80ad:5800:0058:0800:2023:1d71. In addition, a number of agreed-upon simplifications have been standardized for expressing IPv6 addresses:

1. Leading zeros of a block need not be written. In the preceding example, the address could have been written as 5f05:2000:80ad:5800:58:800:2023:1d71.
2. Blocks of all zeros can be omitted and replaced by the notation ::.
 - For example, the IPv6 address 0:0:0:0:0:0:1 can be written more compactly as ::1.
 - Similarly, the address 2001:0db8:0:0:0:0:0:2 can be written more compactly as 2001:db8::2.
 - To avoid ambiguities, the :: notation may be used only once in an IPv6 address
3. **IPv4-mapped IPv6 address.** The block immediately preceding the IPv4 portion of the address has the value ffff and the remaining part of the address is formatted using dotted-quad. For example, the IPv6 address ::ffff:10.0.0.1 represents the IPv4 address 10.0.0.1. This is called an **IPv4-mapped IPv6 address**.
4. **IPv4-compatible IPv6 address.** The low-order 32 bits of the IPv6 address can be written using dotted-quad notation. The IPv6 address ::0102:f001 is therefore equivalent to the address ::1.2.240.1.

The colon delimiter in an IPv6 address may be confused with another separator such as the colon used between an IP address and a port number. In such circumstances, bracket characters, [and], are used to surround the IPv6 address. The following URL is an example:

```
http://[2001:0db8:85a3:08d3:1319:8a2e:0370:7344]:443/
```

The flexibility provided by [RFC4291] resulted in unnecessary confusion due to the ability to represent the same IPv6 address in multiple ways. To remedy this situation, [RFC5952] imposes some rules to narrow the range of options while remaining compatible with [RFC4291]. They are as follows:

1. Leading zeros must be suppressed (e.g., 2001:0db8::0022 becomes 2001:db8::22).
2. The :: construct must be used to its maximum possible effect (most zeros suppressed) but not for only 16-bit blocks. If multiple blocks contain equallength runs of zeros, the first is replaced with ::.
3. The hexadecimal digits a through f should be represented in lowercase.

Basic IP Address Structure

IPv4 has 2^{32} possible addresses and IPv6 has 2^{128} .

- Most of the IPv4 address space is **unicast** address space, which is IPv4 addresses chunks subdivided down to a single address and used to identify a single network interface of a computer attached to the Internet or to some private intranet.
- Most of the IPv6 address space is not currently being used.

Classful Addressing

Subnet Addressing

Subnet Masks

Variable-Length Subnet Masks (VLSM)

Broadcast Addresses

IPv6 Addresses and Interface Identifiers

CIDR and Aggregation

Prefixes

Aggregation

Special-Use Addresses

Addressing IPv4/IPv6 Translators

Multicast Addresses

IPv4 Multicast Addresses

IPv6 Multicast Addresses

Anycast Addresses

An **anycast** address is a unicast IPv4 or IPv6 address that identifies a different host depending on where in the network it is used. This is accomplished by configuring Internet routers to advertise the same unicast routes from multiple locations in the Internet. Thus, an anycast address refers not to a single host in the Internet, but to the "most appropriate" or "closest" single host that is responding to the anycast address.

Anycast addressing is used most frequently for finding a computer that provides a common service. For example, a datagram sent to an anycast address could be used to find a DNS server ([Chapter 11](#)), a 6to4 gateway that encapsulates IPv6 traffic in IPv4 tunnels, or RPs for multicast routing.

Allocation

Unicast

Multicast

Unicast Address Assignment

Single Provider/No Network/Single Address

Single Provider/Single Network/Single Address

Single Provider/Multiple Networks/Multiple Addresses

Multiple Providers/Multiple Networks/Multiple Addresses (Multihoming)

Transmission Control Protocol (TCP)

TCP is a connection oriented protocol and offers end-to-end packet delivery. It acts as back bone for connection. It exhibits the following key features:

- Transmission Control Protocol (TCP) corresponds to the Transport Layer of OSI Model.
- TCP is a reliable and connection oriented protocol.
- TCP offers:
 - Stream Data Transfer.
 - Reliability.
 - Efficient Flow Control
 - Full-duplex operation.
 - Multiplexing.
- TCP offers connection oriented end-to-end packet delivery.
- TCP ensures reliability by sequencing bytes with a forwarding acknowledgement number that indicates to the destination the next byte the source expect to receive.
- It retransmits the bytes not acknowledged with in specified time period.

TCP Services

TCP offers following services to the processes at the application layer:

- Stream Delivery Service
- Sending and Receiving Buffers
- Bytes and Segments
- Full Duplex Service
- Connection Oriented Service

- Reliable Service

Stream Deliver Service

TCP protocol is stream oriented because it allows the sending process to send data as stream of bytes and the receiving process to obtain data as stream of bytes.

Sending and Receiving Buffers

It may not be possible for sending and receiving process to produce and obtain data at same speed, therefore, TCP needs buffers for storage at sending and receiving ends.

Bytes and Segments

The Transmission Control Protocol (TCP), at transport layer groups the bytes into a packet. This packet is called segment. Before transmission of these packets, these segments are encapsulated into an IP datagram.

Full Duplex Service

Transmitting the data in duplex mode means flow of data in both the directions at the same time.

Connection Oriented Service

TCP offers connection oriented service in the following manner:

1. TCP of process-1 informs TCP of process – 2 and gets its approval.
2. TCP of process – 1 and TCP of process – 2 and exchange data in both the two directions.
3. After completing the data exchange, when buffers on both sides are empty, the two TCP's destroy their buffers.

Reliable Service

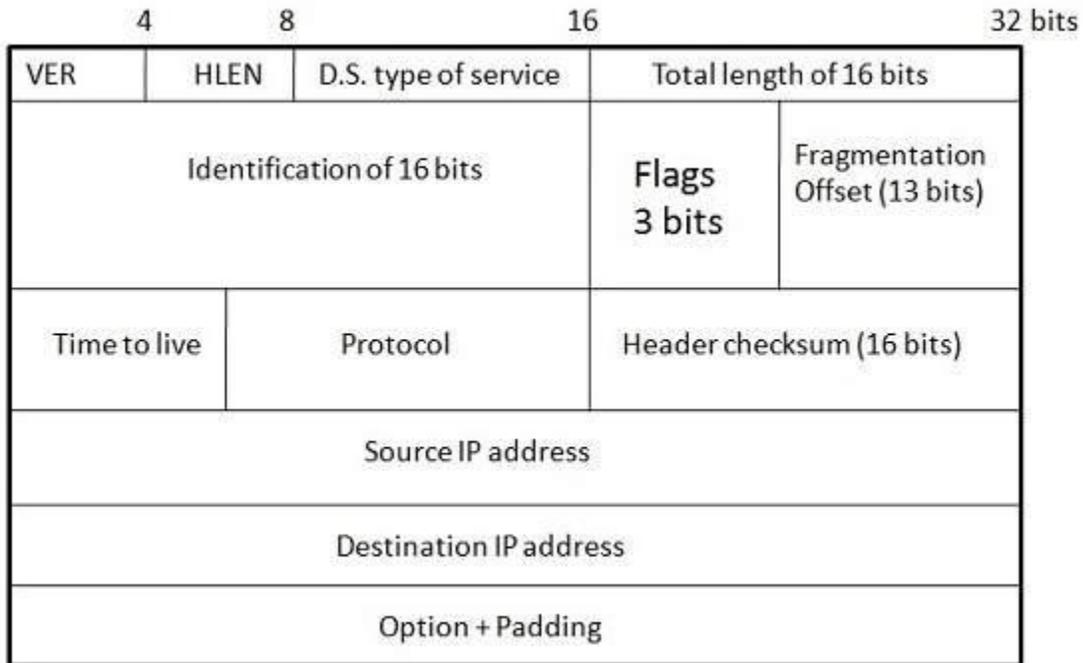
For sake of reliability, TCP uses acknowledgement mechanism.

Internet Protocol (IP)

Internet Protocol is **connectionless** and **unreliable** protocol. It ensures no guarantee of successfully transmission of data.

In order to make it reliable, it must be paired with reliable protocol such as TCP at the transport layer.

Internet protocol transmits the data in form of a datagram as shown in the following diagram:



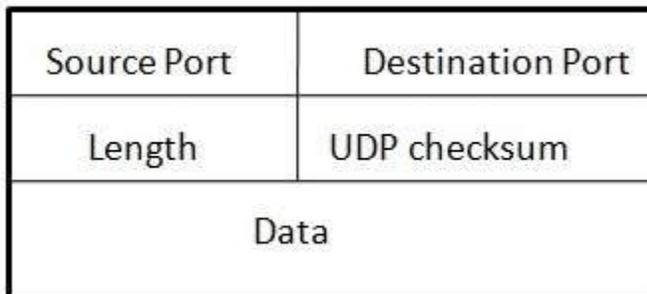
Points to remember:

- The length of datagram is variable.
- The Datagram is divided into two parts: **header** and **data**.
- The length of header is 20 to 60 bytes.
- The header contains information for routing and delivery of the packet.

User Datagram Protocol (UDP)

Like IP, UDP is connectionless and unreliable protocol. It doesn't require making a connection with the host to exchange data. Since UDP is unreliable protocol, there is no mechanism for ensuring that data sent is received.

UDP transmits the data in form of a datagram. The UDP datagram consists of five parts as shown in the following diagram:



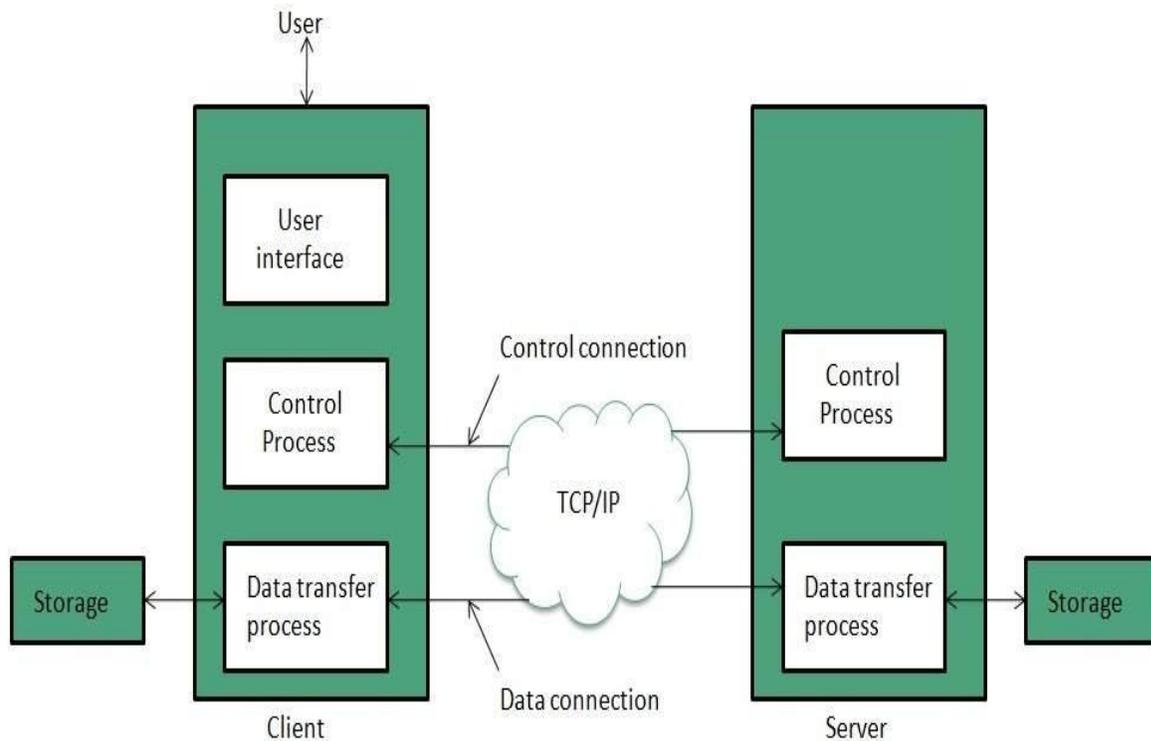
Points to remember:

- UDP is used by the application that typically transmit small amount of data at one time.
- UDP provides protocol port used i.e. UDP message contains both source and destination port number, that makes it possible for UDP software at the destination to deliver the message to correct application program.

File Transfer Protocol (FTP)

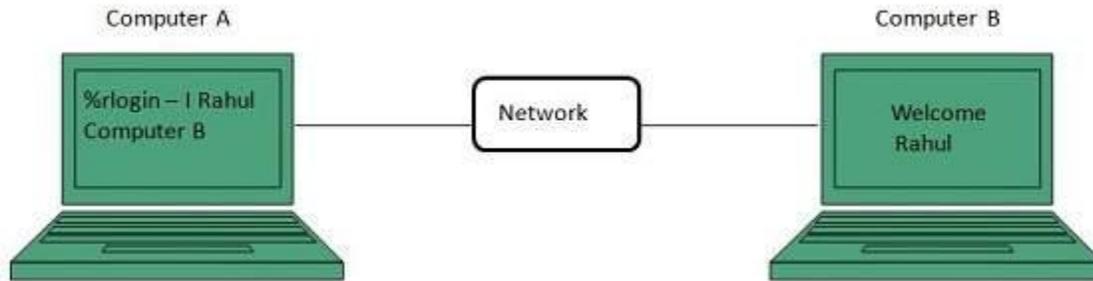
FTP is used to copy files from one host to another. FTP offers the mechanism for the same in following manner:

- FTP creates two processes such as Control Process and Data Transfer Process at both ends i.e. at client as well as at server.
- FTP establishes two different connections: one is for data transfer and other is for control information.
- **Control connection** is made between **control processes** while **Data Connection** is made between
- FTP uses **port 21** for the control connection and **Port 20** for the data connection.



Telnet

Telnet is a protocol used to log in to remote computer on the internet. There are a number of Telnet clients having user friendly user interface. The following diagram shows a person is logged in to computer A, and from there, he remote logged into computer B.



Hyper Text Transfer Protocol (HTTP)

HTTP is a communication protocol. It defines mechanism for communication between browser and the web server. It is also called request and response protocol because the communication between browser and server takes place in request and response pairs.

HTTP Request

HTTP request comprises of lines which contains:

- Request line
- Header Fields
- Message body

Key Points

- The first line i.e. the **Request line** specifies the request method i.e. **Get** or **Post**.
- The second line specifies the header which indicates the domain name of the server from where index.htm is retrieved.

HTTP Response

Like HTTP request, HTTP response also has certain structure. HTTP response contains:

- Status line
- Headers
- Message body

Email

Email is a service which allows us to send the message in electronic mode over the internet. It offers an efficient, inexpensive and real time mean of distributing information among people.

E-Mail Address

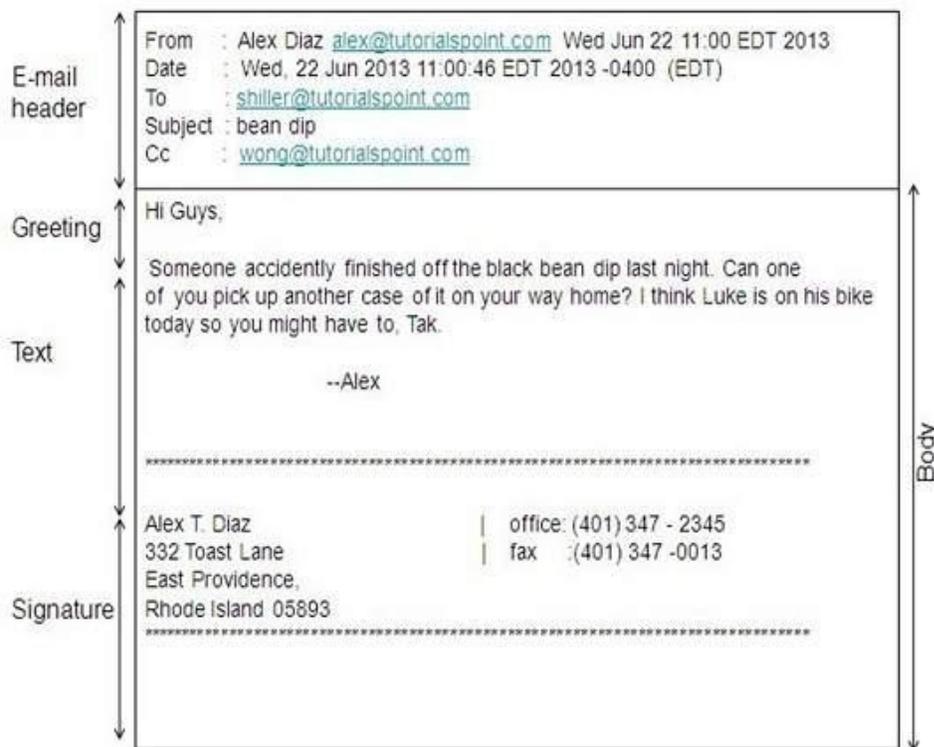
Each user of email is assigned a unique name for his email account. This name is known as E-mail address. Different users can send and receive messages according to the e-mail address.

E-mail is generally of the form `username@domainname`. For example, `webmaster@tutorialspoint.com` is an e-mail address where `webmaster` is username and `tutorialspoint.com` is domain name.

- The username and the domain name are separated by @ (**at**) symbol.
- E-mail addresses are not case sensitive.
- Spaces are not allowed in e-mail address.

E-mail Message Components

E-mail message comprises of different components: E-mail Header, Greeting, Text, and Signature. These components are described in the following diagram:



E-mail Header

The first five lines of an E-mail message is called E-mail header. The header part comprises of following fields:

- From
- Date
- To
- Subject
- CC
- BCC

From

The **From** field indicates the sender's address i.e. who sent the e-mail.

Date

The **Date** field indicates the date when the e-mail was sent.

To

The **To** field indicates the recipient's address i.e. to whom the e-mail is sent.

Subject

The **Subject** field indicates the purpose of e-mail. It should be precise and to the point.

CC

CC stands for Carbon copy. It includes those recipient addresses whom we want to keep informed but not exactly the intended recipient.

BCC

BCC stands for Black Carbon Copy. It is used when we do not want one or more of the recipients to know that someone else was copied on the message.

Greeting

Greeting is the opening of the actual message. Eg. Hi Sir or Hi Guys etc.

Text

It represents the actual content of the message.

Signature

This is the final part of an e-mail message. It includes Name of Sender, Address, and Contact Number.

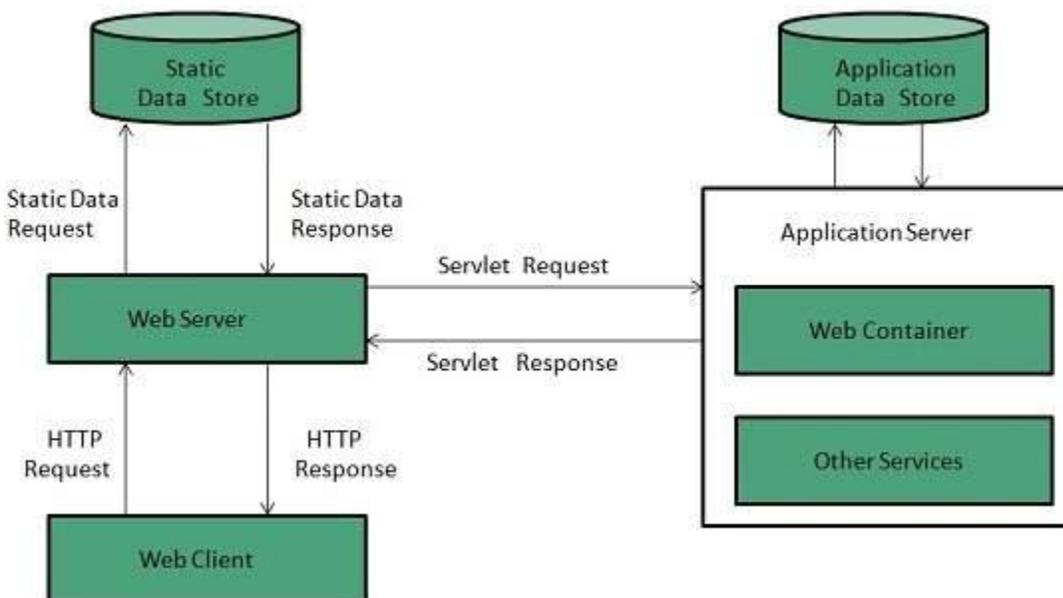
Web server is a computer where the web content is stored. Basically web server is used to host the web sites but there exists other web servers also such as gaming, storage, FTP, email etc.

Web site is collection of web pages while web server is a software that respond to the request for web resources.

Web Server Working

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

- Sending the file to the client associated with the requested URL.
- Generating response by invoking a script and communicating with database



Key Points

- When client sends request for a web page, the web server search for the requested page if requested page is found then it will send it to client with an HTTP response.
- If the requested web page is not found, web server will the send an **HTTP response:Error 404 Not found.**
- If client has requested for some other resources then the web server will contact to the application server and data store to construct the HTTP response.

Architecture

Web Server Architecture follows the following two approaches:

1. Concurrent Approach
2. Single-Process-Event-Driven Approach.

Concurrent Approach

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

- Multi-process
- Multi-threaded
- Hybrid method.

Multi-processing

In this a single process (parent process) initiates several single-threaded child processes and distribute incoming requests to these child processes. Each of the child processes are responsible for handling single request.

It is the responsibility of parent process to monitor the load and decide if processes should be killed or forked.

Multi-threaded

Unlike Multi-process, it creates multiple single-threaded process.

Hybrid

It is combination of above two approaches. In this approach multiple process are created and each process initiates multiple threads. Each of the threads handles one connection. Using multiple threads in single process results in less load on system resources.

Examples

Following table describes the most leading web servers available today:

| S.N. | Web Server Descriptino |
|------|--|
| 1 | Apache HTTP Server This is the most popular web server in the world developed by the Apache Software Foundation. Apache web server is an open source software and can be installed on almost all operating systems including Linux, UNIX, Windows, FreeBSD, Mac OS X and more. About 60% of the web server machines run the Apache Web Server. |
| 2. | Internet Information Services (IIS) The Internet Information Server (IIS) is a high performance Web Server from Microsoft. This web server runs on Windows NT/2000 and 2003 platforms (and may be on upcoming new Windows version also). IIS comes bundled with Windows NT/2000 and 2003; Because IIS is tightly integrated with the operating system so it is relatively easy to administer it. |

| | |
|----|--|
| 3. | <p>Lighttpd The lighttpd, pronounced lighty is also a free web server that is distributed with the FreeBSD operating system. This open source web server is fast, secure and consumes much less CPU power. Lighttpd can also run on Windows, Mac OS X, Linux and Solaris operating systems.</p> |
| 4. | <p>Sun Java System Web Server This web server from Sun Microsystems is suited for medium and large web sites. Though the server is free it is not open source. It however, runs on Windows, Linux and UNIX platforms. The Sun Java System web server supports various languages, scripts and technologies required for Web 2.0 such as JSP, Java Servlets, PHP, Perl, Python, and Ruby on Rails, ASP and Coldfusion etc.</p> |
| 5. | <p>Jigsaw Server Jigsaw (W3C's Server) comes from the World Wide Web Consortium. It is open source and free and can run on various platforms like Linux, UNIX, Windows, and Mac OS X Free BSD etc. Jigsaw has been written in Java and can run CGI scripts and PHP programs.</p> |

Proxy server is an intermediary server between client and the internet. Proxy servers offers the following basic functionalities:

- Firewall and network data filtering.
- Network connection sharing
- Data caching

Proxy servers allow to hide, conceal and make your network id anonymous by hiding your IP address.

Purpose of Proxy Servers

Following are the reasons to use proxy servers:

- Monitoring and Filtering
- Improving performance
- Translation
- Accessing services anonymously
- Security

Monitoring and Filtering

Proxy servers allow us to do several kind of filtering such as:

- Content Filtering
- Filtering encrypted data

- Bypass filters
- Logging and eavesdropping

Improving performance

It fasten the service by process of retrieving content from the cache which was saved when previous request was made by the client.

Translation

It helps to customize the source site for local users by excluding source content or substituting source content with original local content. In this the traffic from the global users is routed to the source website through Translation proxy.

Accessing services anonymously

In this the destination server receives the request from the anonymizing proxy server and thus does not receive information about the end user.

Security

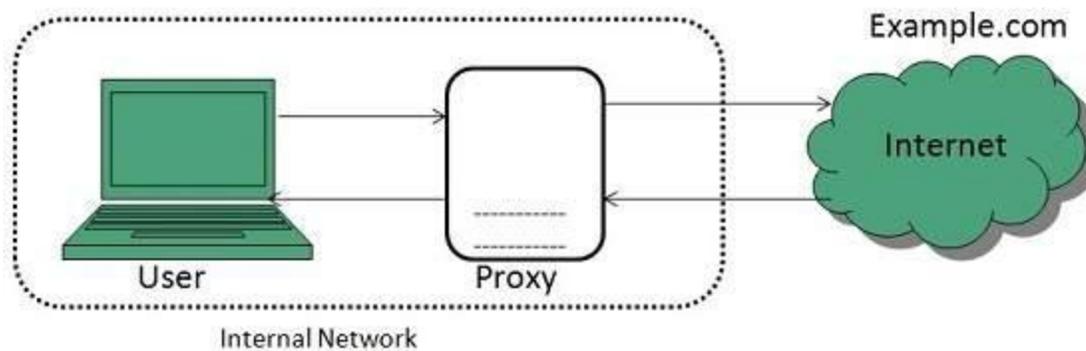
Since the proxy server hides the identity of the user hence it protects from spam and the hacker attacks.

Type of Proxies

Following table briefly describes the type of proxies:

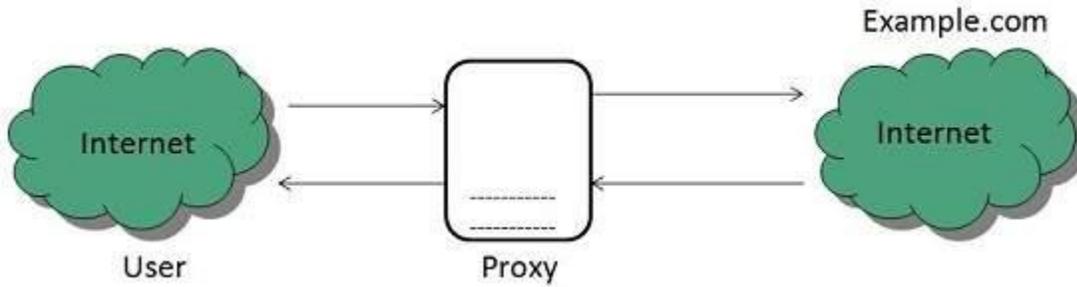
Forward Proxies

In this the client requests its internal network server to forward to the internet.



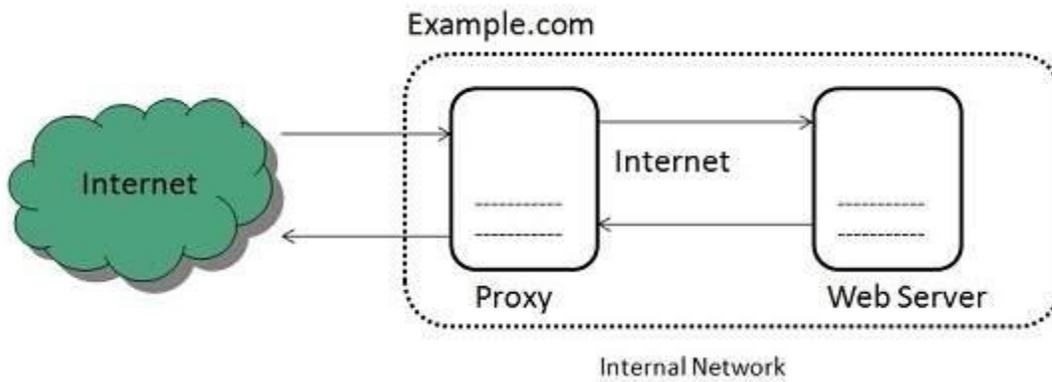
Open Proxies

Open Proxies helps the clients to conceal their IP address while browsing the web.



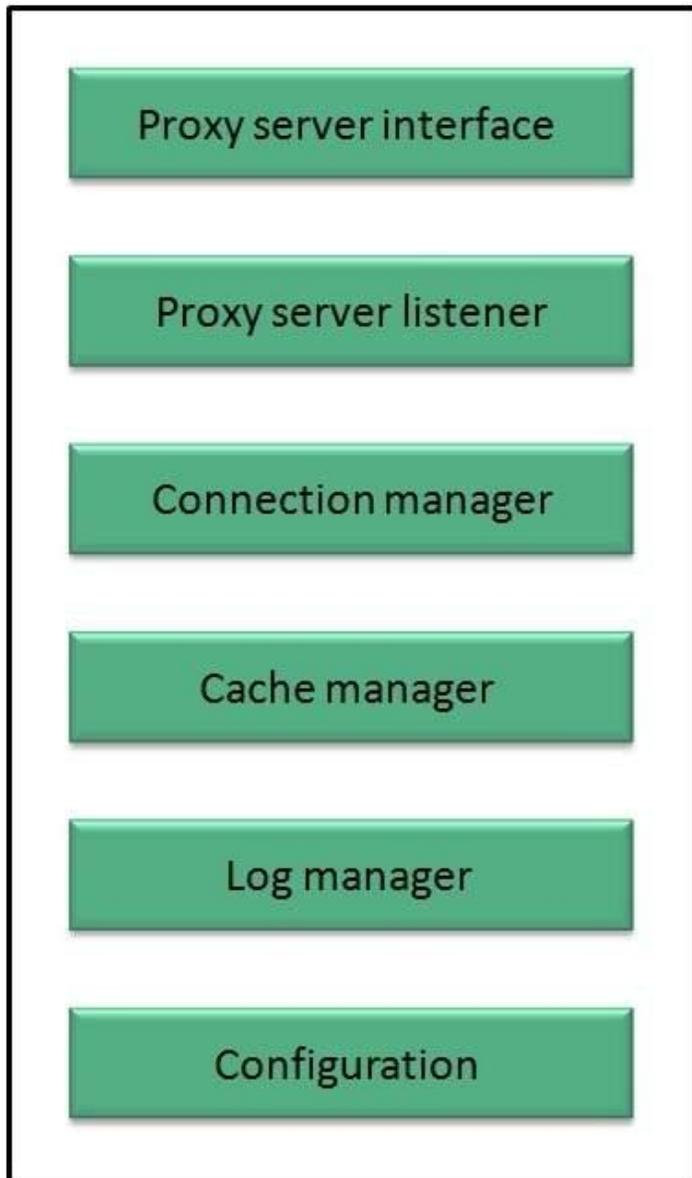
Reverse Proxies

In this the requests are forwarded to one or more proxy servers and the response from the proxy server is retrieved as if it came directly from the original Server.



Architecture

The proxy server architecture is divided into several modules as shown in the following diagram:



Proxy user interface

This module controls and manages the user interface and provides an easy to use graphical interface, window and a menu to the end user. This menu offers the following functionalities:

- Start proxy
- Stop proxy
- Exit
- Blocking URL
- Blocking client
- Manage log

- Manage cache
- Modify configuration

Proxy server listener

It is the port where new request from the client browser is listened. This module also performs blocking of clients from the list given by the user.

Connection Manager

It contains the main functionality of the proxy server. It performs the following functions:

- It contains the main functionality of the proxy server. It performs the following functions:
- Read request from header of the client.
- Parse the URL and determine whether the URL is blocked or not.
- Generate connection to the web server.
- Read the reply from the web server.
- If no copy of page is found in the cache then download the page from web server else will check its last modified date from the reply header and accordingly will read from the cache or server from the web.
- Then it will also check whether caching is allowed or not and accordingly will cache the page.

Cache Manager

This module is responsible for storing, deleting, clearing and searching of web pages in the cache.

Log Manager

This module is responsible for viewing, clearing and updating the logs.

Configuration

This module helps to create configuration settings which in turn let other modules to perform desired configurations such as caching.

What is HTML?

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
 - HTML describes the structure of Web pages using markup
 - HTML elements are the building blocks of HTML pages
 - HTML elements are represented by tags
 - HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
 - Browsers do not display the HTML tags, but use them to render the content of the page
-

A Simple HTML Document

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

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

</body>
</html>
```

Example Explained

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

HTML Tags

HTML tags are element names surrounded by angle brackets:

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

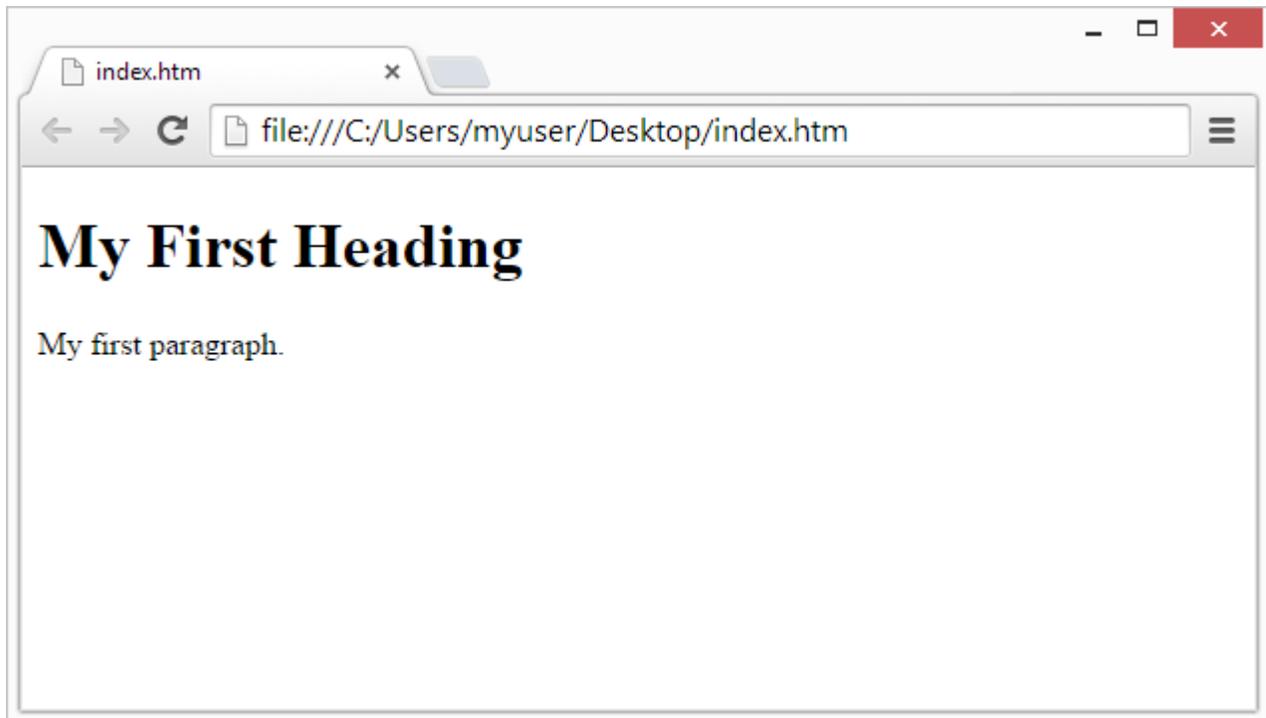
- HTML tags normally come **in pairs** like <p> and </p>
- The first tag in a pair is the **start tag**, the second tag is the **end tag**
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

Tip: The start tag is also called the **opening tag**, and the end tag the **closing tag**.

Web Browsers

The purpose of a web browser (Chrome, IE, Firefox, Safari) is to read HTML documents and display them.

The browser does not display the HTML tags, but uses them to determine how to display the document:



HTML Page Structure

Below is a visualization of an HTML page structure:

```
<html>
<head>
<title>Page title</title>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
</body>
</html>
```

Note: Only the content inside the `<body>` section (the white area above) is displayed in a browser.

The `<!DOCTYPE>` Declaration

The `<!DOCTYPE>` declaration represents the document type, and helps browsers to display web pages correctly.

It must only appear once, at the top of the page (before any HTML tags).

The `<!DOCTYPE>` declaration is not case sensitive.

The `<!DOCTYPE>` declaration for HTML is:

```
<!DOCTYPE html>
```

HTML Versions

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

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

HTML Text Formatting

Text Formatting

This text is bold

This text is italic

This is _{subscript} and ^{superscript}

HTML Formatting Elements

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

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

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

Formatting elements were designed to display special types of text:

- `` - Bold text
 - `` - Important text
 - `<i>` - Italic text
 - `` - Emphasized text
 - `<mark>` - Marked text
 - `<small>` - Small text
 - `` - Deleted text
 - `<ins>` - Inserted text
 - `<sub>` - Subscript text
 - `<sup>` - Superscript text
-

HTML `` and `` Elements

The HTML `` element defines **bold** text, without any extra importance.

Example

```
<b>This text is bold</b>
```

The HTML `` element defines **strong** text, with added semantic "strong" importance.

Example

``This text is strong``

HTML `<i>` and `` Elements

The HTML `<i>` element defines *italic* text, without any extra importance.

Example

`<i>`This text is italic`</i>`

The HTML `` element defines *emphasized* text, with added semantic importance.

Example

``This text is emphasized``

Note: Browsers display `` as ``, and `` as `<i>`. However, there is a difference in the meaning of these tags: `` and `<i>` defines bold and italic text, but `` and `` means that the text is "important".

HTML `<small>` Element

The HTML `<small>` element defines smaller text:

Example

`<h2>`HTML `<small>`Small`</small>` Formatting`</h2>`

HTML `<mark>` Element

The HTML `<mark>` element defines marked or highlighted text:

Example

`<h2>`HTML `<mark>`Marked`</mark>` Formatting`</h2>`

HTML `` Element

The HTML `` element defines ~~deleted~~ (removed) text.

Example

`<p>My favorite color is blue red.</p>`

HTML `<ins>` Element

The HTML `<ins>` element defines [inserted](#) (added) text.

Example

`<p>My favorite color is red.</p>`

HTML `<sub>` Element

The HTML `<sub>` element defines subscripted text.

Example

`<p>This is subscripted text.</p>`

HTML `<sup>` Element

The HTML `<sup>` element defines superscripted text.

Example

`<p>This is superscripted text.</p>`

HTML Text Formatting Elements

| Tag | Description |
|--------------------------------|-------------------------|
| | Defines bold text |
| | Defines emphasized text |
| <i> | Defines italic text |
| <small> | Defines smaller text |
| | Defines important text |

- [<sub>](#) Defines subscripted text
- [<sup>](#) Defines superscripted text
- [<ins>](#) Defines inserted text
- [](#) Defines deleted text
- [<mark>](#) Defines marked/highlighted text

HTML Quotation and Citation Elements

Quotation

Here is a quote from WWF's website:

For 50 years, WWF has been protecting the future of nature. The world's leading conservation organization, WWF works in 100 countries and is supported by 1.2 million members in the United States and close to 5 million globally.

HTML <q> for Short Quotations

The HTML <q> element defines a short quotation.

Browsers usually insert quotation marks around the <q> element.

Example

<p>WWF's goal is to: <q>Build a future where people live in harmony with nature.</q></p>

HTML <blockquote> for Quotations

The HTML <blockquote> element defines a section that is quoted from another source.

Browsers usually indent <blockquote> elements.

Example

<p>Here is a quote from WWF's website:</p>
<blockquote cite="http://www.worldwildlife.org/who/index.html">

For 50 years, WWF has been protecting the future of nature. The world's leading conservation organization, WWF works in 100 countries and is supported by 1.2 million members in the United States and close to 5 million globally.

</blockquote>

HTML <abbr> for Abbreviations

The HTML <abbr> element defines an abbreviation or an acronym.

Marking abbreviations can give useful information to browsers, translation systems and search-engines.

Example

<p>The <abbr title="World Health Organization">WHO</abbr> was founded in 1948.</p>

HTML <address> for Contact Information

The HTML <address> element defines contact information (author/owner) of a document or an article.

The <address> element is usually displayed in italic. Most browsers will add a line break before and after the element.

Example

```
<address>
Written by John Doe.<br>
Visit us at:<br>
Example.com<br>
Box 564, Disneyland<br>
USA
</address>
```

HTML <cite> for Work Title

The HTML <cite> element defines the title of a work.

Browsers usually display <cite> elements in italic.

Example

<p><cite>The Scream</cite> by Edvard Munch. Painted in 1893.</p>

HTML <bdo> for Bi-Directional Override

The HTML <bdo> element defines bi-directional override.

The <bdo> element is used to override the current text direction:

Example

<bdo dir="rtl">This text will be written from right to left</bdo>

HTML Quotation and Citation Elements

| Tag | Description |
|------------------------------------|--|
| <abbr> | Defines an abbreviation or acronym |
| <address> | Defines contact information for the author/owner of a document |
| <bdo> | Defines the text direction |
| <blockquote> | Defines a section that is quoted from another source |
| <cite> | Defines the title of a work |
| <q> | Defines a short inline quotation |

HTML Links

Links are found in nearly all web pages. Links allow users to click their way from page to page.

HTML Links - Hyperlinks

HTML links are hyperlinks.

You can click on a link and jump to another document.

When you move the mouse over a link, the mouse arrow will turn into a little hand.

Note: A link does not have to be text. It can be an image or any other HTML element.

HTML Links - Syntax

In HTML, links are defined with the `<a>` tag:

```
<a href="url">link text</a>
```

Example

```
<a href="http://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

The **href** attribute specifies the destination address (`http://www.w3schools.com/html/`) of the link.

The **link text** is the visible part (Visit our HTML tutorial).

Clicking on the link text will send you to the specified address.

Note: Without a forward slash on subfolder addresses, you might generate two requests to the server. Many servers will automatically add a forward slash to the address, and then create a new request.

Local Links

The example above used an absolute URL (A full web address).

A local link (link to the same web site) is specified with a relative URL (without `http://www....`).

Example

```
<a href="html_images.asp">HTML Images</a>
```

HTML Link Colors

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

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

You can change the default colors, by using styles:

Example

```
<style>
a:link {color:green; background-color:transparent; text-decoration:none}
a:visited {color:pink; background-color:transparent; text-decoration:none}
a:hover {color:red; background-color:transparent; text-decoration:underline}
a:active {color:yellow; background-color:transparent; text-decoration:underline}
</style>
```

HTML Links - The target Attribute

The **target** attribute specifies where to open the linked document.

The target attribute can have one of the following values:

- `_blank` - Opens the linked document in a new window or tab
- `_self` - Opens the linked document in the same window/tab as it was clicked (this is default)
- `_parent` - Opens the linked document in the parent frame
- `_top` - Opens the linked document in the full body of the window
- `framename` - Opens the linked document in a named frame

This example will open the linked document in a new browser window/tab:

Example

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

Tip: If your webpage is locked in a frame, you can use `target="_top"` to break out of the frame:

Example

```
<a href="http://www.w3schools.com/html/" target="_top">HTML5 tutorial!</a>
```

HTML Links - Image as Link

It is common to use images as links:

Example

```
<a href="default.asp">
  
</a>
```

Note: `border:0;` is added to prevent IE9 (and earlier) from displaying a border around the image (when the image is a link).

HTML Links - Create a Bookmark

HTML bookmarks are used to allow readers to jump to specific parts of a Web page.

Bookmarks can be useful if your webpage is very long.

To make a bookmark, you must first create the bookmark, and then add a link to it.

When the link is clicked, the page will scroll to the location with the bookmark.

Example

First, create a bookmark with the id attribute:

```
<h2 id="tips">Useful Tips Section</h2>
```

Then, add a link to the bookmark ("Useful Tips Section"), from within the same page:

```
<a href="#tips">Visit the Useful Tips Section</a>
```

Or, add a link to the bookmark ("Useful Tips Section"), from another page:

Example

```
<a href="html_tips.html#tips">Visit the Useful Tips Section</a>
```

External Paths

External pages can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a web page:

Example

```
<a href="http://www.w3schools.com/html/default.asp">HTML tutorial</a>
```

This example links to a page located in the html folder on the current web site:

Example

```
<a href="/html/default.asp">HTML tutorial</a>
```

This example links to a page located in the same folder as the current page:

Example

```
<a href="default.asp">HTML tutorial</a>
```

You can read more about file paths in the chapter [HTML File Paths](#).

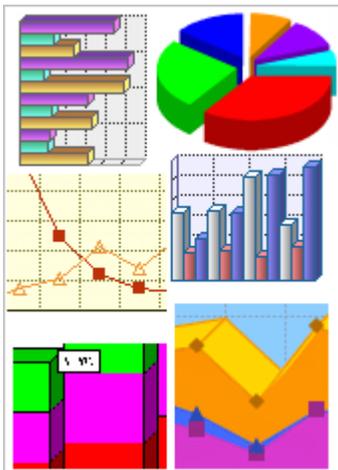
HTML Images

JPG Images



GIF Images

PNG Images



Example

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

html>

```
<body>

<h2>Spectacular                               Mountain</h2>


</body>
</html>
```

HTML Images Syntax

In HTML, images are defined with the `` tag.

The `` tag is empty, it contains attributes only, and does not have a closing tag.

The `src` attribute specifies the URL (web address) of the image:

```

```

The alt Attribute

The `alt` attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the `src` attribute, or if the user uses a screen reader).

If a browser cannot find an image, it will display the value of the `alt` attribute:

Example

```

```

The `alt` attribute is required. A web page will not validate correctly without it.

HTML Screen Readers

A screen reader is a software program that reads the HTML code, converts the text, and allows the user to "listen" to the content. Screen readers are useful for people who are blind, visually impaired, or learning disabled.

Image Size - Width and Height

You can use the **style** attribute to specify the width and height of an image.

The values are specified in pixels (use px after the value):

Example

```

```

Alternatively, you can use the **width** and **height** attributes. Here, the values are specified in pixels by default:

Example

```

```

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

Width and Height, or Style?

Both the width, height, and style attributes are valid in HTML5.

However, we suggest using the style attribute. It prevents internal or external styles sheets from changing the original size of images:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
img {
width:100%;
}
</style>
</head>
<body>




</body>
</html>
```

Images in Another Folder

If not specified, the browser expects to find the image in the same folder as the web page.

However, it is common to store images in a sub-folder. You must then include the folder name in the src attribute:

Example

```

```

Images on Another Server

Some web sites store their images on image servers.

Actually, you can access images from any web address in the world:

Example

```

```

You can read more about file paths in the chapter [HTML File Paths](#).

Animated Images

The GIF standard allows animated images:

Example

```

```

Note that the syntax of inserting animated images is no different from non-animated images.

Using an Image as a Link

To use an image as a link, simply nest the tag inside the <a> tag:

Example

```
<a href="default.asp">
  
</a>
```

Note: border:0; is added to prevent IE9 (and earlier) from displaying a border around the image (when the image is a link).

Image Floating

Use the CSS **float** property to let the image float to the right or to the left of a text:

Example

```
<p>
The image will float to the right of the text.</p>
```

```
<p>
The image will float to the left of the text.</p>
```

Image Maps

Use the <map> tag to define an image-map. An image-map is an image with clickable areas.

The name attribute of the <map> tag is associated with the 's usemap attribute and creates a relationship between the image and the map.

The <map> tag contains a number of <area> tags, that defines the clickable areas in the image-map:

Example

```

<map name="planetmap">
  <area shape="rect" coords="0,0,82,126" alt="Sun" href="sun.htm">
  <area shape="circle" coords="90,58,3" alt="Mercury" href="mercur.htm">
  <area shape="circle" coords="124,58,8" alt="Venus" href="venus.htm">
</map>
```

Chapter Sum

- **img** element to define an image
- Use the HTML **src** attribute to define the URL of the image
- Use the HTML **alt** attribute to define an alternate text for an image, if it cannot be displayed
- Use the HTML **width** and **height** attributes to define the size of the image
- Use the CSS **width** and **height** properties to define the size of the image (alternatively)
- Use the CSS **float** property to let the image float
- Use the HTML **<map>** element to define an image-map
- Use the HTML **<area>** element to define the clickable areas in the image-map
- Use the HTML ****'s element **usemap** attribute to point to an image-map

HTML Image Tags

| Tag | Description |
|------------------------------|--|
| | Defines an image |
| <map> | Defines an image-map |
| <area> | Defines a clickable area inside an image-map |

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

The HTML tables are created using the **<table>** tag in which the **<tr>** tag is used to create table rows and **<td>** tag is used to create data cells.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Tables</title>
</head>
<body>
<table border="1">
<tr>
```

```
<td>Row 1, Column 1</td>
<td>Row 1, Column 2</td>
</tr>
<tr>
<td>Row 2, Column 1</td>
<td>Row 2, Column 2</td>
</tr>
</table>
</body>
</html>
```

This will produce following result:

| | |
|-----------------|-----------------|
| Row 1, Column 1 | Row 1, Column 2 |
| Row 2, Column 1 | Row 2, Column 2 |

Here **border** is an attribute of <table> tag and it is used to put a border across all the cells. If you do not need a border then you can use border="0".

Table Heading

Table heading can be defined using <th> tag. This tag will be put to replace <td> tag, which is used to represent actual data cell. Normally you will put your top row as table heading as shown below, otherwise you can use <th> element in any row.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Header</title>
</head>
<body>
<table border="1">
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>
<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>
```

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

This will produce following result:

| Name | Salary |
|-----------------|--------|
| Ramesh Raman | 5000 |
| Shabbir Hussein | 7000 |

Cellpadding and Cellspacing Attributes

There are two attributes called *cellpadding* and *cellspacing* which you will use to adjust the white space in your table cells. The cellspacing attribute defines the width of the border, while cellpadding represents the distance between cell borders and the content within a cell.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Cellpadding</title>
</head>
<body>
<table border="1" cellpadding="5" cellspacing="5">
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>
<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>
</table>
</body>
</html>
```

This will produce following result:

| Name | Salary |
|------|--------|
|------|--------|

| | |
|-----------------|------|
| Ramesh Raman | 5000 |
| Shabbir Hussein | 7000 |

Colspan and Rowspan Attributes

You will use **colspan** attribute if you want to merge two or more columns into a single column. Similar way you will use **rowspan** if you want to merge two or more rows.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Colspan/Rowspan</title>
</head>
<body>
<table border="1">
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
<tr><td rowspan="2">Row 1 Cell 1</td><td>Row 1 Cell 2</td><td>Row 1 Cell
3</td></tr>
<tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
<tr><td colspan="3">Row 3 Cell 1</td></tr>
</table>
</body>
</html>
```

This will produce following result:

| Column 1 | Column 2 | Column 3 |
|--------------|--------------|--------------|
| Row 1 Cell 1 | Row 1 Cell 2 | Row 1 Cell 3 |
| | Row 2 Cell 2 | Row 2 Cell 3 |
| Row 3 Cell 1 | | |

Tables Backgrounds

You can set table background using one of the following two ways:

- **bgcolor** attribute - You can set background color for whole table or just for one cell.
- **background** attribute - You can set background image for whole table or just

for one cell.

You can also set border color also using **bordercolor** attribute.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Background</title>
</head>
<body>
<table border="1" bordercolor="green" bgcolor="yellow">
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
<tr><td rowspan="2">Row 1 Cell 1</td><td>Row 1 Cell 2</td><td>Row 1 Cell
3</td></tr>
<tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
<tr><td colspan="3">Row 3 Cell 1</td></tr>
</table>
</body>
</html>
```

This will produce following result:

| Column 1 | Column 2 | Column 3 |
|--------------|--------------|--------------|
| Row 1 Cell 1 | Row 1 Cell 2 | Row 1 Cell 3 |
| | Row 2 Cell 2 | Row 2 Cell 3 |
| Row 3 Cell 1 | | |

Here is an example of using **background** attribute. Here we will use an image available in /images directory.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Background</title>
</head>
<body>
<table border="1" bordercolor="green" background="/images/test.png">
<tr>
<th>Column 1</th>
```

```

<th>Column 2</th>
<th>Column 3</th>
</tr>
<tr><td rowspan="2">Row 1 Cell 1</td><td>Row 1 Cell 2</td><td>Row 1 Cell
3</td></tr>
<tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
<tr><td colspan="3">Row 3 Cell 1</td></tr>
</table>
</body>
</html>

```

This will produce following result. Here background image did not apply to table's header.

| Column 1 | Column 2 | Column 3 |
|--------------|--------------|--------------|
| Row 1 Cell 1 | Row 1 Cell 2 | Row 1 Cell 3 |
| | Row 2 Cell 2 | Row 2 Cell 3 |
| Row 3 Cell 1 | | |

Table Height and Width

You can set a table width and height using **width** and **height** attributes. You can specify table width or height in terms of pixels or in terms of percentage of available screen area.

Example

```

<!DOCTYPE html>
<html>
<head>
<title>HTML Table Width/Height</title>
</head>
<body>
<table border="1" width="400" height="150">
<tr>
<td>Row 1, Column 1</td>
<td>Row 1, Column 2</td>
</tr>
<tr>
<td>Row 2, Column 1</td>
<td>Row 2, Column 2</td>
</tr>
</table>
</body>
</html>

```

This will produce following result:

| | |
|-----------------|-----------------|
| Row 1, Column 1 | Row 1, Column 2 |
| Row 2, Column 1 | Row 2, Column 2 |

Table Caption

The **caption** tag will serve as a title or explanation for the table and it shows up at the top of the table. This tag is deprecated in newer version of HTML/XHTML.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table Caption</title>
</head>
<body>
<table border="1" width="100%">
<caption>This is the caption</caption>
<tr>
<td>row 1, column 1</td><td>row 1, columnn 2</td>
</tr>
<tr>
<td>row 2, column 1</td><td>row 2, columnn 2</td>
</tr>
</table>
</body>
</html>
```

This will produce following result:

| | |
|---------------------|------------------|
| This is the caption | |
| row 1, column 1 | row 1, columnn 2 |
| row 2, column 1 | row 2, columnn 2 |

Table Header, Body, and Footer

Tables can be divided into three portions: a header, a body, and a foot. The head and foot are rather similar to headers and footers in a word-processed document that remain the same for every page, while the body is the main content holder of the table.

The three elements for separating the head, body, and foot of a table are:

- **<thead>** - to create a separate table header.
- **<tbody>** - to indicate the main body of the table.
- **<tfoot>** - to create a separate table footer.

A table may contain several `<tbody>` elements to indicate different *pages* or groups of data. But it is notable that `<thead>` and `<tfoot>` tags should appear before `<tbody>`

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table</title>
</head>
<body>
<table border="1" width="100%">
<thead>
<tr>
<td colspan="4">This is the head of the table</td>
</tr>
</thead>
<tfoot>
<tr>
<td colspan="4">This is the foot of the table</td>
</tr>
</tfoot>
<tbody>
<tr>
<td>Cell 1</td>
<td>Cell 2</td>
<td>Cell 3</td>
<td>Cell 4</td>
</tr>
</tbody>
</table>
</body>
</html>
```

This will produce following result:

| | | | |
|-------------------------------|--------|--------|--------|
| This is the head of the table | | | |
| This is the foot of the table | | | |
| Cell 1 | Cell 2 | Cell 3 | Cell 4 |

Nested Tables

You can use one table inside another table. Not only tables you can use almost all the tags inside table data tag <td>.

Example

Following is the example of using another table and other tags inside a table cell.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table</title>
</head>
<body>
<table border="1" width="100%">
<tr>
<td>
<table border="1" width="100%">
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>
<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>
</table>
</td>
</tr>
</table>
</body>
</html>
```

This will produce following result:

| Name | Salary |
|-----------------|---------------|
| Ramesh Raman | 5000 |
| Shabbir Hussein | 7000 |

HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements. Lists may contain:

- **** - An unordered list. This will list items using plain bullets.
- **** - An ordered list. This will use different schemes of numbers to list your items.
- **<dl>** - A definition list. This arranges your items in the same way as they are arranged in a dictionary.

HTML Unordered Lists

An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML **** tag. Each item in the list is marked with a bullet.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
<ul>
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
<li>Radish</li>
</ul>
</body>
</html>
```

This will produce following result:

- Beetroot
- Ginger
- Potato
- Radish

The type Attribute

You can use **type** attribute for **** tag to specify the type of bullet you like. By default it is a disc. Following are the possible options:

```
<ul type="square">
<ul type="disc">
<ul type="circle">
```

Example

Following is an example where we used `<ul type="square">`

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
  <ul type="square">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ul>
</body>
</html>
```

This will produce following result:

- Beetroot
- Ginger
- Potato
- Radish

Example

Following is an example where we used `<ul type="disc">` :

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
  <ul type="disc">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ul>
</body>
</html>
```

This will produce following result:

- Beetroot
- Ginger

- Potato
- Radish

Example

Following is an example where we used `<ul type="circle">` :

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
  <ul type="circle">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ul>
</body>
</html>
```

This will produce following result:

- Beetroot
- Ginger
- Potato
- Radish

HTML Ordered Lists

If you are required to put your items in a numbered list instead of bulleted then HTML ordered list will be used. This list is created by using `` tag. The numbering starts at one and is incremented by one for each successive ordered list element tagged with ``.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
<ol>
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
```

```
<li>Radish</li>
</ol>
</body>
</html>
```

This will produce following result:

1. Beetroot
2. Ginger
3. Potato
4. Radish

The type Attribute

You can use **type** attribute for tag to specify the type of numbering you like. By default it is a number. Following are the possible options:

```
<ol type="1"> - Default-Case Numerals.
<ol type="I"> - Upper-Case Numerals.
<ol type="i"> - Lower-Case Numerals.
<ol type="a"> - Lower-Case Letters.
<ol type="A"> - Upper-Case Letters.
```

Example

Following is an example where we used <ol type="1">

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
  <ol type="1">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ol>
</body>
</html>
```

This will produce following result:

1. Beetroot
2. Ginger
3. Potato

4. Radish

Example

Following is an example where we used `<ol type="I">`

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
  <ol type="I">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ol>
</body>
</html>
```

This will produce following result:

- I. Beetroot
- II. Ginger
- III. Potato
- IV. Radish

Example

Following is an example where we used `<ol type="i">`

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
  <ol type="i">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ol>
</body>
</html>
```

This will produce following result:

- i. Beetroot
- ii. Ginger
- iii. Potato
- iv. Radish

Example

Following is an example where we used `<ol type="A">`

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
  <ol type="A">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ol>
</body>
</html>
```

This will produce following result:

- A. Beetroot
- B. Ginger
- C. Potato
- D. Radish

Example

Following is an example where we used `<ol type="a">`

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
  <ol type="a">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
```

```
<li>Radish</li>
</ol>
</body>
</html>
```

This will produce following result:

- a. Beetroot
- b. Ginger
- c. Potato
- d. Radish

The start Attribute

You can use **start** attribute for `` tag to specify the starting point of numbering you need. Following are the possible options:

```
<ol type="1" start="4"> - Numerals starts with 4.
<ol type="I" start="4"> - Numerals starts with IV.
<ol type="i" start="4"> - Numerals starts with iv.
<ol type="a" start="4"> - Letters starts with d.
<ol type="A" start="4"> - Letters starts with D.
```

Example

Following is an example where we used `<ol type="i" start="4" >`

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
  <ol type="i" start="4">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ol>
</body>
</html>
```

This will produce following result:

- iv. Beetroot
- v. Ginger
- vi. Potato

vii. Radish

HTML Definition Lists

HTML and XHTML support a list style which is called **definition lists** where entries are listed like in a dictionary or encyclopedia. The definition list is the ideal way to present a glossary, list of terms, or other name/value list.

Definition List makes use of following three tags.

- `<dl>` - Defines the start of the list
- `<dt>` - A term
- `<dd>` - Term definition
- `</dl>` - Defines the end of the list

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Definition List</title>
</head>
<body>
<dl>
<dt><b>HTML</b></dt>
<dd>This stands for Hyper Text Markup Language</dd>
<dt><b>HTTP</b></dt>
<dd>This stands for Hyper Text Transfer Protocol</dd>
</dl>
</body>
</html>
```

This will produce following result:

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.

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 following result:

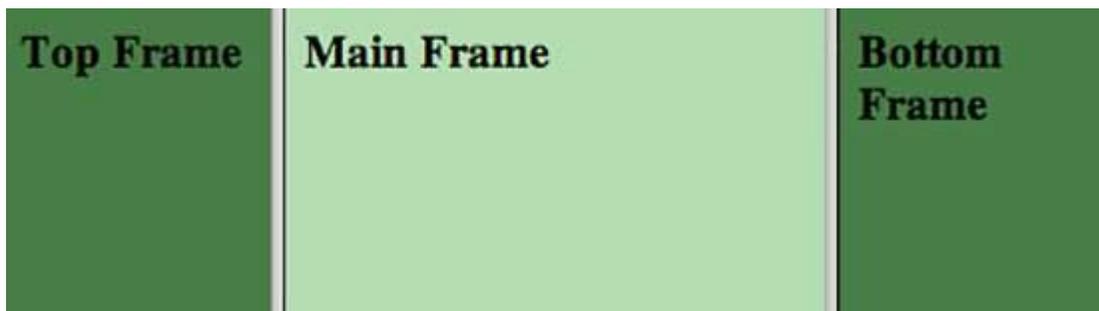


Example

Let's put 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" />
</frameset>
</html>
```

This will produce following result:



The <frameset> Tag Attributes

Following are important attributes of the <frameset> tag:

| Attribute | Description |
|------------------|--|
| cols | <p>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 four ways:</p> <ul style="list-style-type: none">• Absolute values in pixels. For example to create three vertical frames, use <i>cols="100, 500,100"</i>.• A percentage of the browser window. For example to create three vertical frames, use <i>cols="10%, 80%,10%"</i>.• Using a wildcard symbol. For example to create three vertical frames, use <i>cols="10%, *,10%"</i>. In this case wildcard takes remainder of the window.• As relative widths of the browser window. For example to create three vertical frames, use <i>cols="3*,2*,1*"</i>. 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. |
| rows | <p>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 <i>rows="10%, 90%"</i>. You can specify the height of each row in the same way as explained above for columns.</p> |
| border | <p>This attribute specifies the width of the border of each frame in pixels. For example <i>border="5"</i>. A value of zero means no border.</p> |
| frameborder | <p>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 <i>frameborder="0"</i> specifies no border.</p> |
| framespacing | <p>This attribute specifies the amount of space between frames in a frameset. This can take any integer value. For example <i>framespacing="10"</i> means there should be 10 pixels spacing between each frames.</p> |

The <frame> Tag Attributes

Following are important attributes of <frame> tag:

| Attribute | Description |
|------------------|---|
| src | <p>This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, <i>src="/html/top_frame.htm"</i> will load an HTML file available in html directory.</p> |
| name | <p>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.</p> |
| frameborder | <p>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</p> |

take values either 1 (yes) or 0 (no).

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"`.

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"`.

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"`.

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.

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" />
</frameset>
<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 menubar 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 menubar, we have mentioned target frame as **main_page**, so whenever you click any of the links in menubar, available link will open in main_page.

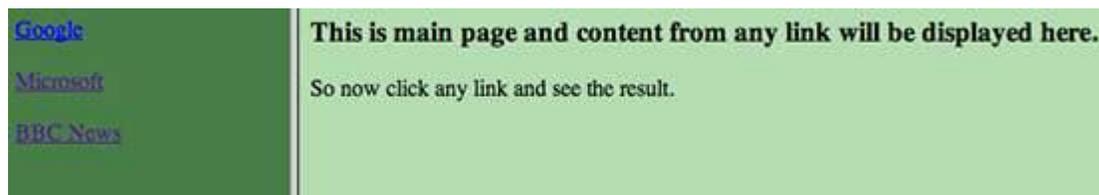
Following is the content of menu.htm file

```
<!DOCTYPE html>
<html>
<body bgcolor="#4a7d49">
<a href="https://www.google.com" target="main_page">Google</a>
<br /><br />
<a href="https://www.microsoft.com" target="main_page">Microsoft</a>
<br /><br />
<a href="https://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 *target* attribute can also take one of the following values:

Option

Description

`_self` Loads the page into the current frame.
`_blank` Loads a page into a new browser window.opening a new window.
`_parent` Loads the page into the parent window, which in the case of a single frameset is the main browser window.
`_top` Loads the page into the browser window, replacing any current frames.
`targetframe` Loads the page into a named targetframe.

HTML Form

HTML Forms are required when you want to collect some data from the site visitor. For example during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML **<form>** tag is used to create an HTML form and it has following syntax:

```
<form action="Script URL" method="GET|POST">  
  form elements like input, textarea etc.  
</form>
```

Form Attributes

Apart from common attributes, following is a list of the most frequently used form attributes:

| Attribute | Description |
|---------------------|--|
| <code>action</code> | Backend script ready to process your passed data. |
| <code>method</code> | Method to be used to upload data. The most frequently used are GET and POST methods. |
| <code>target</code> | Specify the target window or frame where the result of the script will be displayed. It takes values like <code>_blank</code> , <code>_self</code> , <code>_parent</code> etc. |

You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are:

- enctype
- **application/x-www-form-urlencoded** - This is the standard method most forms use in simple scenarios.
 - **multipart/form-data** - This is used when you want to upload binary data in the form of files like image, word file etc.

Note: You can refer to [Perl & CGI](#) for a detail on how form data upload works.

HTML Form Controls

There are different types of form controls that you can use to collect data using HTML form:

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Clickable Buttons
- Submit and Reset Button

Text Input Controls

There are three types of text input used on forms:

- **Single-line text input controls** - This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML **<input>** tag.
- **Password input controls** - This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML **<input>** tag.
- **Multi-line text input controls** - This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML **<textarea>** tag.

Single-line text input controls

This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML **<input>** tag.

Example

Here is a basic example of a single-line text input used to take first name and last name:

```
<!DOCTYPE html>
<html>
<head>
<title>Text Input Control</title>
</head>
<body>
<form >
First name: <input type="text" name="first_name" />
<br>
Last name: <input type="text" name="last_name" />
</form>
</body>
</html>
```

This will produce following result:

First name:

Last name:

Attributes

Following is the list of attributes for <input> tag for creating text field.

| Attribute | Description |
|-----------|--|
| type | Indicates the type of input control and for text input control it will be set to text . |
| name | Used to give a name to the control which is sent to the server to be recognized and get the value. |
| value | This can be used to provide an initial value inside the control. |
| size | Allows to specify the width of the text-input control in terms of characters. |
| maxlength | Allows to specify the maximum number of characters a user can enter into the text box. |

Password input controls

This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input> tag but type attribute is set to **password**.

Example

Here is a basic example of a single-line password input used to take user password:

```
<!DOCTYPE html>
```

```

<html>
<head>
<title>Password Input Control</title>
</head>
<body>
<form >
User ID : <input type="text" name="user_id" />
<br>
Password: <input type="password" name="password" />
</form>
</body>
</html>

```

This will produce following result:

User ID :

Password:

Attributes

Following is the list of attributes for <input> tag for creating password field.

| Attribute | Description |
|-----------|--|
| type | Indicates the type of input control and for password input control it will be set to password . |
| name | Used to give a name to the control which is sent to the server to be recognized and get the value. |
| value | This can be used to provide an initial value inside the control. |
| size | Allows to specify the width of the text-input control in terms of characters. |
| maxlength | Allows to specify the maximum number of characters a user can enter into the text box. |

Multiple-Line Text Input Controls

This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

Example

Here is a basic example of a multi-line text input used to take item description:

```

<!DOCTYPE html>
<html>

```

```

<head>
<title>Multiple-Line Input Control</title>
</head>
<body>
<form>
Description : <br />
<textarea rows="5" cols="50" name="description">
Enter description here...
</textarea>
</form>
</body>
</html>

```

This will produce following result:

Description :



Attributes

Following is the list of attributes for <textarea> tag.

| Attribute | Description |
|-----------|--|
| name | Used to give a name to the control which is sent to the server to be recognized and get the value. |
| rows | Indicates the number of rows of text area box. |
| cols | Indicates the number of columns of text area box |

Checkbox Control

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **checkbox**.

Example

Here is an example HTML code for a form with two checkboxes:

```

<!DOCTYPE html>
<html>

```

```
<head>
<title>Checkbox Control</title>
</head>
<body>
<form>
<input type="checkbox" name="maths" value="on"> Maths
<input type="checkbox" name="physics" value="on"> Physics
</form>
</body>
</html>
```

This will produce following result:

Maths Physics

Attributes

Following is the list of attributes for <checkbox> tag.

| Attribute | Description |
|-----------|--|
| type | Indicates the type of input control and for checkbox input control it will be set to checkbox . |
| name | Used to give a name to the control which is sent to the server to be recognized and get the value. |
| value | The value that will be used if the checkbox is selected. |
| checked | Set to <i>checked</i> if you want to select it by default. |

Radio Button Control

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **radio**.

Example

Here is example HTML code for a form with two radio buttons:

```
<!DOCTYPE html>
<html>
<head>
<title>Radio Box Control</title>
</head>
<body>
<form>
```

```
<input type="radio" name="subject" value="maths"> Maths
<input type="radio" name="subject" value="physics"> Physics
</form>
</body>
</html>
```

This will produce following result:

Maths Physics

Attributes

Following is the list of attributes for radio button.

| Attribute | Description |
|-----------|---|
| type | Indicates the type of input control and for checkbox input control it will be set to radio . |
| name | Used to give a name to the control which is sent to the server to be recognized and get the value. |
| value | The value that will be used if the radio box is selected. |
| checked | Set to <i>checked</i> if you want to select it by default. |

Select Box Control

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

Example

Here is example HTML code for a form with one drop down box

```
<!DOCTYPE html>
<html>
<head>
<title>Select Box Control</title>
</head>
<body>
<form>
<select name="dropdown">
<option value="Maths" selected>Maths</option>
<option value="Physics">Physics</option>
</select>
</form>
```

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

This will produce following result:

Attributes

Following is the list of important attributes of <select> tag:

| Attribute | Description |
|-----------|--|
| name | Used to give a name to the control which is sent to the server to be recognized and get the value. |
| size | This can be used to present a scrolling list box. |
| multiple | If set to "multiple" then allows a user to select multiple items from the menu. |

Following is the list of important attributes of <option> tag:

| Attribute | Description |
|-----------|--|
| value | The value that will be used if an option in the select box box is selected. |
| selected | Specifies that this option should be the initially selected value when the page loads. |
| label | An alternative way of labeling options |

File Upload Box

If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the <input> element but type attribute is set to **file**.

Example

Here is example HTML code for a form with one file upload box:

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<input type="file" name="fileupload" accept="image/*" />
</form>
```

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

This will produce following result:

Attributes

Following is the list of important attributes of file upload box:

| Attribute | Description |
|-----------|--|
| name | Used to give a name to the control which is sent to the server to be recognized and get the value. |
| accept | Specifies the types of files that the server accepts. |

Button Controls

There are various ways in HTML to create clickable buttons. You can also create a clickable button using `<input>` tag by setting its type attribute to **button**. The type attribute can take the following values:

| Type | Description |
|--------|--|
| submit | This creates a button that automatically submits a form. |
| reset | This creates a button that automatically resets form controls to their initial values. |
| button | This creates a button that is used to trigger a client-side script when the user clicks that button. |
| image | This creates a clickable button but we can use an image as background of the button. |

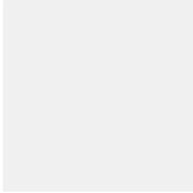
Example

Here is example HTML code for a form with three types of buttons:

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset" value="Reset" />
<input type="button" name="ok" value="OK" />
<input type="image" name="imagebutton" src="/html/images/logo.png" />
```

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

This will produce following result:



Hidden Form Controls

Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page has be displayed next based on the passed current page.

Example

Here is example HTML code to show the usage of hidden control:

```
<!DOCTYPE html>
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<p>This is page 10</p>
<input type="hidden" name="pagename" value="10" />
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset" value="Reset" />
</form>
</body>
</html>
```

This will produce following result:

This is page 10

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 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 following result:

Hello, World!

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 following result:

This is red

This is thick

This is green

This is thick and green

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 following result:

This is red

This is thick

This is green

This is thick and green

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>
```

This will produce following result:

This is red

This is thick

This is green

This is thick and green

CSS - Text

Advertisements

[Previous Page](#)

[Next Page](#)

This chapter teaches you how to manipulate text using CSS properties. You can set following text properties of an element –

- The **color** property is used to set the color of a text.
- The **direction** property is used to set the text direction.
- The **letter-spacing** property is used to add or subtract space between the letters that make up a word.
- The **word-spacing** property is used to add or subtract space between the words of a sentence.
- The **text-indent** property is used to indent the text of a paragraph.
- The **text-align** property is used to align the text of a document.
- The **text-decoration** property is used to underline, overline, and strikethrough text.
- The **text-transform** property is used to capitalize text or convert text to uppercase or lowercase letters.
- The **white-space** property is used to control the flow and formatting of text.
- The **text-shadow** property is used to set the text shadow around a text.

Set the Text Color

The following example demonstrates how to set the text color. Possible value could be any color name in any valid format.

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

```
<p style="color:red;">
  This text will be written in red.
</p>
</body>
</html>
```

It will produce the following result –

Set the Text Direction

The following example demonstrates how to set the direction of a text. Possible values are *ltr* or *rtl*.

```
<html>
<head>
</head>
<body>
  <p style="direction:rtl;">
    This text will be rendered from right to left
  </p>
</body>
</html>
```

It will produce the following result –

Set the Space between Characters

The following example demonstrates how to set the space between characters. Possible values are *normal* or a number specifying space..

```
<html>
<head>
</head>
<body>
  <p style="letter-spacing:5px;">
    This text is having space between letters.
  </p>
</body>
</html>
```

It will produce the following result –

Set the Space between Words

The following example demonstrates how to set the space between words. Possible values are *normal* or a number specifying space.

```
<html>
  <head>
</head>
  <body>
    <p style="word-spacing:5px;">
      This text is having space between words.
    </p>
  </body>
</html>
```

It will produce the following result –

Set the Text Indent

The following example demonstrates how to indent the first line of a paragraph. Possible values are *%* or *a number specifying indent space*.

```
<html>
  <head>
</head>
  <body>
    <p style="text-indent:1cm;">
      This text will have first line indented by 1cm and this line will remain at
      its actual position this is done by CSS text-indent property.
    </p>
  </body>
</html>
```

It will produce the following result –

Set the Text Alignment

The following example demonstrates how to align a text. Possible values are *left*, *right*, *center*, *justify*.

```
<html>
  <head>
</head>
  <body>
    <p style="text-align:right;">
      This will be right aligned.
    </p>

    <p style="text-align:center;">
      This will be center aligned.
    </p>
```

```
<p style="text-align:left;">
This will be left aligned.
</p>
```

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

It will produce the following result –

Decorating the Text

The following example demonstrates how to decorate a text. Possible values are *none*, *underline*, *overline*, *line-through*, *blink*.

```
<html>
<head>
</head>
<body>
  <p style="text-decoration:underline;">
    This will be underlined
  </p>

  <p style="text-decoration:line-through;">
    This will be striked through.
  </p>

  <p style="text-decoration:overline;">
    This will have a over line.
  </p>

  <p style="text-decoration:blink;">
    This text will have blinking effect
  </p>
</body>
</html>
```

It will produce the following result &minnus;

Set the Text Cases

The following example demonstrates how to set the cases for a text. Possible values are *none*, *capitalize*, *uppercase*, *lowercase*.

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

```
<body>
  <p style="text-transform:capitalize;">
    This will be capitalized
  </p>

  <p style="text-transform:uppercase;">
    This will be in uppercase
  </p>

  <p style="text-transform:lowercase;">
    This will be in lowercase
  </p>
</body>
```

```
</html>
```

It will produce the following result:

Set the White Space between Text

The following example demonstrates how white space inside an element is handled. Possible values are *normal*, *pre*, *nowrap*.

```
<html>
  <head>
  </head>
  <body>
    <p style="white-space:pre;">
      This text has a line break and the white-space pre setting tells the browser to honor
      it just like the HTML pre tag.</p>
  </body>
</html>
```

It will produce the following result –

Set the Text Shadow

The following example demonstrates how to set the shadow around a text. This may not be supported by all the browsers.

```
<html>
  <head>
  </head>
  <body>
    <p style="text-shadow:4px 4px 8px blue;">
      If your browser supports the CSS text-shadow property, this text will have a blue shadow.
```

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

It will produce the following result –

CSS - Using Images

Advertisements

[Previous Page](#)

[Next Page](#)

Images play an important role in any webpage. Though it is not recommended to include a lot of images, but it is still important to use good images wherever required.

CSS plays a good role to control image display. You can set the following image properties using CSS.

- The **border** property is used to set the width of an image border.
- The **height** property is used to set the height of an image.
- The **width** property is used to set the width of an image.
- The **-moz-opacity** property is used to set the opacity of an image.

The Image Border Property

The *border* property of an image is used to set the width of an image border. This property can have a value in length or in %.

A width of zero pixels means no border.

Here is the example –

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

```

<br />

</body>
</html>
```

It will produce the following result –

The Image Height Property

The *height* property of an image is used to set the height of an image. This property can have a value in length or in %. While giving value in %, it applies it in respect of the box in which an image is available.

Here is an example –

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

<br />

</body>
</html>
```

It will produce the following result –

The Image Width Property

The *width* property of an image is used to set the width of an image. This property can have a value in length or in %. While giving value in %, it applies it in respect of the box in which an image is available.

Here is an example –

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

<br />

</body>
</html>
```

It will produce the following result –

The `-moz-opacity` Property

The `-moz-opacity` property of an image is used to set the opacity of an image. This property is used to create a transparent image in Mozilla. IE uses **`filter:alpha(opacity=x)`** to create transparent images.

In Mozilla (`-moz-opacity:x`) `x` can be a value from 0.0 - 1.0. A lower value makes the element more transparent (The same thing goes for the CSS3-valid syntax `opacity:x`).

In IE (`filter:alpha(opacity=x)`) `x` can be a value from 0 - 100. A lower value makes the element more transparent.

Here is an example –

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

It will produce the following result –

CSS - Tables

Advertisements

[Previous Page](#)

[Next Page](#)

This tutorial will teach you how to set different properties of an HTML table using CSS. You can set following properties of a table –

- The **`border-collapse`** specifies whether the browser should control the appearance of the adjacent borders that touch each other or whether each cell should maintain its style.
- The **`border-spacing`** specifies the width that should appear between table cells.
- The **`caption-side`** captions are presented in the `<caption>` element. By default, these are rendered above the table in the document. You use the `caption-side` property to control the placement of the table caption.

- The **empty-cells** specifies whether the border should be shown if a cell is empty.
- The **table-layout** allows browsers to speed up layout of a table by using the first width properties it comes across for the rest of a column rather than having to load the whole table before rendering it.

Now, we will see how to use these properties with examples.

The border-collapse Property:

This property can have two values *collapse* and *separate*. The following example uses both the values:

```
<html>
<head>

  <style type="text/css">
    table.one {border-collapse:collapse;}
    table.two {border-collapse:separate;}
    td.a {
      border-style:dotted;
      border-width:3px;
      border-color:#000000;
      padding: 10px;
    }
    td.b {
      border-style:solid;
      border-width:3px;
      border-color:#333333;
      padding:10px;
    }
  </style>

</head>
<body>

  <table class="one">
    <caption>Collapse Border Example</caption>
    <tr><td class="a"> Cell A Collapse Example</td></tr>
    <tr><td class="b"> Cell B Collapse Example</td></tr>
  </table>
  <br />

  <table class="two">
    <caption>Separate Border Example</caption>
    <tr><td class="a"> Cell A Separate Example</td></tr>
    <tr><td class="b"> Cell B Separate Example</td></tr>
  </table>
```

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

It will produce the following result –

The border-spacing Property

The border-spacing property specifies the distance that separates adjacent cells' borders. It can take either one or two values; these should be units of length.

If you provide one value, it will apply to both vertical and horizontal borders. Or you can specify two values, in which case, the first refers to the horizontal spacing and the second to the vertical spacing –

NOTE – Unfortunately, this property does not work in Netscape 7 or IE 6.

```
<style type="text/css">
  /* If you provide one value */
  table.example {border-spacing:10px;}
  /* This is how you can provide two values */
  table.example {border-spacing:10px; 15px;}
</style>
```

Now let's modify the previous example and see the effect –

```
<html>
<head>

  <style type="text/css">
    table.one {
      border-collapse:separate;
      width:400px;
      border-spacing:10px;
    }
    table.two {
      border-collapse:separate;
      width:400px;
      border-spacing:10px 50px;
    }
  </style>

</head>
<body>

  <table class="one" border="1">
    <caption>Separate Border Example with border-spacing</caption>
    <tr><td> Cell A Collapse Example</td></tr>
```

```
<tr><td> Cell B Collapse Example</td></tr>
</table>
<br />
```

```
<table class="two" border="1">
  <caption>Separate Border Example with border-spacing</caption>
  <tr><td> Cell A Separate Example</td></tr>
  <tr><td> Cell B Separate Example</td></tr>
</table>
```

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

It will produce the following result –

The caption-side Property

The caption-side property allows you to specify where the content of a <caption> element should be placed in relationship to the table. The table that follows lists the possible values.

This property can have one of the four values *top*, *bottom*, *left* or *right*. The following example uses each value.

NOTE: These properties may not work with your IE Browser.

```
<html>
<head>

  <style type="text/css">
    caption.top {caption-side:top}
    caption.bottom {caption-side:bottom}
    caption.left {caption-side:left}
    caption.right {caption-side:right}
  </style>

</head>
<body>

  <table style="width:400px; border:1px solid black;">
    <caption class="top">
      This caption will appear at the top
    </caption>
    <tr><td > Cell A</td></tr>
    <tr><td > Cell B</td></tr>
  </table>
<br />
```

```

<table style="width:400px; border:1px solid black;">
  <caption class="bottom">
    This caption will appear at the bottom
  </caption>
  <tr><td > Cell A</td></tr>
  <tr><td > Cell B</td></tr>
</table>
<br />

```

```

<table style="width:400px; border:1px solid black;">
  <caption class="left">
    This caption will appear at the left
  </caption>
  <tr><td > Cell A</td></tr>
  <tr><td > Cell B</td></tr>
</table>
<br />

```

```

<table style="width:400px; border:1px solid black;">
  <caption class="right">
    This caption will appear at the right
  </caption>
  <tr><td > Cell A</td></tr>
  <tr><td > Cell B</td></tr>
</table>

```

```

</body>
</html>

```

It will produce the following result –

The empty-cells Property

The empty-cells property indicates whether a cell without any content should have a border displayed.

This property can have one of the three values - *show*, *hide* or *inherit*.

Here is the empty-cells property used to hide borders of empty cells in the <table> element.

```

<html>
<head>

  <style type="text/css">
    table.empty{
      width:350px;
      border-collapse:separate;
      empty-cells:hide;
    }
  </style>

```

```

    }
    td.empty{
        padding:5px;
        border-style:solid;
        border-width:1px;
        border-color:#999999;
    }
</style>

</head>
<body>

<table class="empty">
<tr>
    <th></th>
    <th>Title one</th>
    <th>Title two</th>
</tr>

<tr>
    <th>Row Title</th>
    <td class="empty">value</td>
    <td class="empty">value</td>
</tr>

<tr>
    <th>Row Title</th>
    <td class="empty">value</td>
    <td class="empty"></td>
</tr>
</table>

</body>
</html>

```

It will produce the following result –

The table-layout Property

The table-layout property is supposed to help you control how a browser should render or lay out a table.

This property can have one of the three values: *fixed*, *auto* or *inherit*.

The following example shows the difference between these properties.

NOTE – This property is not supported by many browsers so do not rely on this property.

[Previous Page](#)

[Next Page](#)

The *border* properties allow you to specify how the border of the box representing an element should look. There are three properties of a border you can change:

- The **border-color** specifies the color of a border.
- The **border-style** specifies whether a border should be solid, dashed line, double line, or one of the other possible values.
- The **border-width** specifies the width of a border.

Now, we will see how to use these properties with examples.

The border-color Property

The border-color property allows you to change the color of the border surrounding an element. You can individually change the color of the bottom, left, top and right sides of an element's border using the properties –

- **border-bottom-color** changes the color of bottom border.
- **border-top-color** changes the color of top border.
- **border-left-color** changes the color of left border.
- **border-right-color** changes the color of right border.

The following example shows the effect of all these properties –

```
<html>
<head>

  <style type="text/css">
    p.example1 {
      border: 1px solid;
      border-bottom-color: #009900; /* Green */
      border-top-color: #FF0000; /* Red */
      border-left-color: #330000; /* Black */
      border-right-color: #0000CC; /* Blue */
    }
    p.example2 {
      border: 1px solid;
      border-color: #009900; /* Green */
    }
  </style>

</head>
```

```
<body>

  <p class="example1">
    This example is showing all borders in different colors.
  </p>

  <p class="example2">
    This example is showing all borders in green color only.
  </p>

</body>
</html>
```

It will produce the following result –

The border-style Property

The border-style property allows you to select one of the following styles of border –

- **none:** No border. (Equivalent of border-width:0;)
- **solid:** Border is a single solid line.
- **dotted:** Border is a series of dots.
- **dashed:** Border is a series of short lines.
- **double:** Border is two solid lines.
- **groove:** Border looks as though it is carved into the page.
- **ridge:** Border looks the opposite of groove.
- **inset:** Border makes the box look like it is embedded in the page.
- **outset:** Border makes the box look like it is coming out of the canvas.
- **hidden:** Same as none, except in terms of border-conflict resolution for table elements.

You can individually change the style of the bottom, left, top, and right borders of an element using the following properties –

- **border-bottom-style** changes the style of bottom border.
- **border-top-style** changes the style of top border.
- **border-left-style** changes the style of left border.
- **border-right-style** changes the style of right border.

The following example shows all these border styles –

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

  <body>
    <p style="border-width:4px; border-style:none;">
```

This is a border with none width.

</p>

<p style="border-width:4px; border-style:solid;">

This is a solid border.

</p>

<p style="border-width:4px; border-style:dashed;">

This is a dashed border.

</p>

<p style="border-width:4px; border-style:double;">

This is a double border.

</p>

<p style="border-width:4px; border-style:groove;">

This is a groove border.

</p>

<p style="border-width:4px; border-style:ridge">

This is a ridge border.

</p>

<p style="border-width:4px; border-style:inset;">

This is an inset border.

</p>

<p style="border-width:4px; border-style:outset;">

This is an outset border.

</p>

<p style="border-width:4px; border-style:hidden;">

This is a hidden border.

</p>

<p style="border-width:4px;border-top-style:solid;

border-bottom-style:dashed; border-left-style:groove; border-right-style:double;">

This is a border with four different styles.

</p>

</body>

</html>

It will produce the following result –

The border-width Property –

The border-width property allows you to set the width of an element borders. The value of this property could be either a length in px, pt or cm or it should be set to *thin, medium or thick*.

You can individually change the width of the bottom, top, left, and right borders of an element using the following properties –

- **border-bottom-width** changes the width of bottom border.
- **border-top-width** changes the width of top border.
- **border-left-width** changes the width of left border.
- **border-right-width** changes the width of right border.

The following example shows all these border width –

```
<html>
  <head>
  </head>
  <body>
    <p style="border-width:4px; border-style:solid;">
      This is a solid border whose width is 4px.
    </p>

    <p style="border-width:4pt; border-style:solid;">
      This is a solid border whose width is 4pt.
    </p>

    <p style="border-width:thin; border-style:solid;">
      This is a solid border whose width is thin.
    </p>

    <p style="border-width:medium; border-style:solid;">
      This is a solid border whose width is medium;
    </p>

    <p style="border-width:thick; border-style:solid;">
      This is a solid border whose width is thick.
    </p>

    <p style="border-bottom-width:4px;border-top-width:10px;
      border-left-width: 2px;border-right-width:15px;border-style:solid;">
      This is a a border with four different width.
    </p>
  </body>
</html>
```

It will produce the following result –

Border Properties Using Shorthand

The border property allows you to specify color, style, and width of lines in one property –

The following example shows how to use all the three properties into a single property. This is the most frequently used property to set border around any element.

```
<html>
  <head>
  </head>
  <body>
    <p style="border:4px solid red;">
      This example is showing shorthand property for border.
    </p>
  </body>
</html>
```

It will produce the following result –

Unit III

Java Script

What is JavaScript ?

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as **LiveScript**, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are –

- **Less server interaction** – You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** – They don't have to wait for a page reload to see if they have forgotten to enter something.

- **Increased interactivity** – You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** – You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

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

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

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Development Tools

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here –

- **Microsoft FrontPage** – Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.
- **Macromedia Dreamweaver MX** – Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.
- **Macromedia HomeSite 5** – HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

JavaScript - 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.

[Live Demo](#)

```
<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.

NOTE – Care should be taken while writing variable and function names 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>
```

Enabling JavaScript in Browsers

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. This chapter explains the procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera.

JavaScript in Internet Explorer

Here are simple steps to turn on or turn off JavaScript in your Internet Explorer –

- Follow **Tools** → **Internet Options** from the menu.
- Select **Security** tab from the dialog box.
- Click the **Custom Level** button.
- Scroll down till you find **Scripting** option.
- Select *Enable* radio button under **Active scripting**.
- Finally click OK and come out

To disable JavaScript support in your Internet Explorer, you need to select **Disable** radio button under **Active scripting**.

JavaScript in Firefox

Here are the steps to turn on or turn off JavaScript in Firefox –

- Open a new tab → type **about: config** in the address bar.
- Then you will find the warning dialog. Select **I'll be careful, I promise!**
- Then you will find the list of **configure options** in the browser.
- In the search bar, type **javascript.enabled**.
- There you will find the option to enable or disable javascript by right-clicking on the value of that option → **select toggle**.

If javascript.enabled is true; it converts to false upon clicking **toggle**. If javascript is disabled; it gets enabled upon clicking toggle.

JavaScript in Chrome

Here are the steps to turn on or turn off JavaScript in Chrome –

- Click the Chrome menu at the top right hand corner of your browser.
- Select **Settings**.
- Click **Show advanced settings** at the end of the page.
- Under the **Privacy** section, click the Content settings button.
- In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

JavaScript in Opera

Here are the steps to turn on or turn off JavaScript in Opera –

- Follow **Tools** → **Preferences** from the menu.
- Select **Advanced** option from the dialog box.
- Select **Content** from the listed items.
- Select **Enable JavaScript** checkbox.
- Finally click OK and come out.

To disable JavaScript support in your Opera, you should not select the **Enable JavaScript checkbox**.

Warning for Non-JavaScript Browsers

If you have to do something important using JavaScript, then you can display a warning message to the user using `<noscript>` tags.

You can add a **noscript** block immediately after the script block as follows –

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

  <noscript>
    Sorry...JavaScript is needed to go ahead.
  </noscript>
</body>
</html>
```

Now, if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from `</noscript>` will be displayed on the screen.

JavaScript - Placement in HTML File

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows –

- Script in `<head>...</head>` section.
- Script in `<body>...</body>` section.
- Script in `<body>...</body>` and `<head>...</head>` sections.
- Script in an external file and then include in `<head>...</head>` section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

JavaScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows –

[Live Demo](#)

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

<body>
  <input type = "button" onclick = "sayHello()" value = "Say Hello" />
</body>
</html>
```

This code will produce the following results –

JavaScript in <body>...</body> section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

[Live Demo](#)

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

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

  <p>This is web page body </p>
</body>
</html>
```

This code will produce the following results –

JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows –

[Live Demo](#)

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

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

    <input type = "button" onclick = "sayHello()" value = "Say Hello" />
  </body>
</html>
```

This code will produce the following result –

JavaScript in External File

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using **script** tag and its **src** attribute.

```
<html>
  <head>
    <script type = "text/javascript" src = "filename.js" ></script>
  </head>

  <body>
    .....
  </body>
```

```
</html>
```

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in **filename.js** file and then you can use **sayHello** function in your HTML file after including the filename.js file.

```
function sayHello() {  
    alert("Hello World")  
}
```

JavaScript - Variables

JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types –

- **Numbers**, eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined**, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

Note – JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

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;  
  //-->  
</script>
```

You can also declare multiple variables with the same **var** keyword as follows –

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

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type = "text/javascript">
  <!--
    var name = "Ali";
    var money;
    money = 2000.50;
  //-->
</script>
```

Note – Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

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

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

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

[Live Demo](#)

```
<html>
  <body onload = checkscope();>
    <script type = "text/javascript">
      <!--
        var myVar = "global"; // Declare a global variable
        function checkscope() {
          var myVar = "local"; // Declare a local variable
```

```
        document.write(myVar);
    }
    //-->
</script>
</body>
</html>
```

This produces the following result –

local

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **_123test** is a valid one.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

JavaScript Reserved Words

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

| | | | |
|----------|---------|------------|--------------|
| abstract | else | instanceof | switch |
| boolean | enum | int | synchronized |
| break | export | interface | this |
| byte | extends | long | throw |
| case | false | native | throws |
| catch | final | new | transient |
| char | finally | null | true |

| | | | |
|----------|------------|-----------|----------|
| class | float | package | try |
| const | for | private | typeof |
| continue | function | protected | var |
| debugger | goto | public | void |
| default | if | return | volatile |
| delete | implements | short | while |
| do | import | static | with |
| double | in | super | |

JavaScript - Operators

What is an Operator?

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types of operators.

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

Lets have a look on all operators one by one.

Arithmetic Operators

JavaScript supports the following arithmetic operators –

Assume variable A holds 10 and variable B holds 20, then –

| Sr.No. | Operator & Description |
|--------|------------------------|
|--------|------------------------|

| | |
|---|---|
| 1 | <p>+ (Addition)</p> <p>Adds two operands</p> <p>Ex: A + B will give 30</p> |
| 2 | <p>- (Subtraction)</p> <p>Subtracts the second operand from the first</p> <p>Ex: A - B will give -10</p> |
| 3 | <p>* (Multiplication)</p> <p>Multiply both operands</p> <p>Ex: A * B will give 200</p> |
| 4 | <p>/ (Division)</p> <p>Divide the numerator by the denominator</p> <p>Ex: B / A will give 2</p> |
| 5 | <p>% (Modulus)</p> <p>Outputs the remainder of an integer division</p> <p>Ex: B % A will give 0</p> |
| 6 | <p>++ (Increment)</p> <p>Increases an integer value by one</p> <p>Ex: A++ will give 11</p> |
| 7 | <p>-- (Decrement)</p> <p>Decreases an integer value by one</p> <p>Ex: A-- will give 9</p> |

Note – Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example

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

[Live Demo](#)

```
<html>
```

```
<body>
```

```
<script type = "text/javascript">  
  <!--  
    var a = 33;  
    var b = 10;  
    var c = "Test";  
    var linebreak = "<br />";  
  
    document.write("a + b = ");  
    result = a + b;  
    document.write(result);  
    document.write(linebreak);  
  
    document.write("a - b = ");  
    result = a - b;  
    document.write(result);  
    document.write(linebreak);  
  
    document.write("a / b = ");  
    result = a / b;  
    document.write(result);  
    document.write(linebreak);  
  
    document.write("a % b = ");  
    result = a % b;  
    document.write(result);  
    document.write(linebreak);  
  
    document.write("a + b + c = ");  
    result = a + b + c;  
    document.write(result);  
    document.write(linebreak);  
  
    a = ++a;  
    document.write(++a = );  
    result = ++a;  
    document.write(result);  
    document.write(linebreak);  
  
    b = --b;  
    document.write(--b = );  
    result = --b;  
    document.write(result);  
    document.write(linebreak);  
  //-->  
</script>
```

Set the variables to different values and then try...

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

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
```

Set the variables to different values and then try...

Comparison Operators

JavaScript supports the following comparison operators –

Assume variable A holds 10 and variable B holds 20, then –

| Sr.No. | Operator & Description |
|--------|---|
| 1 | == (Equal) Checks if the value of two operands are equal or not, if yes, then the condition becomes true. Ex: (A == B) is not true. |
| 2 | != (Not Equal) Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. Ex: (A != B) is true. |
| 3 | > (Greater than) Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. Ex: (A > B) is not true. |
| 4 | < (Less than) Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. Ex: (A < B) is true. |

| | |
|---|---|
| 5 | <p>>= (Greater than or Equal to)</p> <p>Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A >= B) is not true.</p> |
| 6 | <p><= (Less than or Equal to)</p> <p>Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A <= B) is true.</p> |

Example

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

[Live Demo](#)

```
<html>
<body>
  <script type = "text/javascript">
    <!--
      var a = 10;
      var b = 20;
      var linebreak = "<br />";

      document.write("(a == b) => ");
      result = (a == b);
      document.write(result);
      document.write(linebreak);

      document.write("(a < b) => ");
      result = (a < b);
      document.write(result);
      document.write(linebreak);

      document.write("(a > b) => ");
      result = (a > b);
      document.write(result);
      document.write(linebreak);

      document.write("(a != b) => ");
      result = (a != b);
      document.write(result);
      document.write(linebreak);

      document.write("(a >= b) => ");
```

```
result = (a >= b);
document.write(result);
document.write(linebreak);

document.write("(a <= b) => ");
result = (a <= b);
document.write(result);
document.write(linebreak);
//-->
</script>
Set the variables to different values and different operators and then try...
</body>
</html>
```

Output

(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b) => true

Set the variables to different values and different operators and then try...

Logical Operators

JavaScript supports the following logical operators –

Assume variable A holds 10 and variable B holds 20, then –

| Sr.No. | Operator & Description |
|--------|---|
| 1 | <p>&& (Logical AND)</p> <p>If both the operands are non-zero, then the condition becomes true.</p> <p>Ex: (A && B) is true.</p> |
| 2 | <p> (Logical OR)</p> <p>If any of the two operands are non-zero, then the condition becomes true.</p> <p>Ex: (A B) is true.</p> |
| 3 | <p>! (Logical NOT)</p> <p>Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.</p> |

Ex: ! (A && B) is false.

Example

Try the following code to learn how to implement Logical Operators in JavaScript.

[Live Demo](#)

```
<html>
<body>
  <script type = "text/javascript">
    <!--
      var a = true;
      var b = false;
      var linebreak = "<br />";

      document.write("(a && b) => ");
      result = (a && b);
      document.write(result);
      document.write(linebreak);

      document.write("(a || b) => ");
      result = (a || b);
      document.write(result);
      document.write(linebreak);

      document.write("!(a && b) => ");
      result = !(a && b);
      document.write(result);
      document.write(linebreak);
    //-->
  </script>
  <p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

Output

(a && b) => false

(a || b) => true

!(a && b) => true

Set the variables to different values and different operators and then try...

Bitwise Operators

JavaScript supports the following bitwise operators –

Assume variable A holds 2 and variable B holds 3, then –

| Sr.No. | Operator & Description |
|--------|--|
| 1 | <p>& (Bitwise AND)</p> <p>It performs a Boolean AND operation on each bit of its integer arguments.</p> <p>Ex: (A & B) is 2.</p> |
| 2 | <p> (BitWise OR)</p> <p>It performs a Boolean OR operation on each bit of its integer arguments.</p> <p>Ex: (A B) is 3.</p> |
| 3 | <p>^ (Bitwise XOR)</p> <p>It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.</p> <p>Ex: (A ^ B) is 1.</p> |
| 4 | <p>~ (Bitwise Not)</p> <p>It is a unary operator and operates by reversing all the bits in the operand.</p> <p>Ex: (~B) is -4.</p> |
| 5 | <p><< (Left Shift)</p> <p>It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.</p> <p>Ex: (A << 1) is 4.</p> |
| 6 | <p>>> (Right Shift)</p> <p>Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.</p> <p>Ex: (A >> 1) is 1.</p> |
| 7 | <p>>>> (Right shift with Zero)</p> <p>This operator is just like the >> operator, except that the bits shifted in on the left are always zero.</p> <p>Ex: (A >>> 1) is 1.</p> |

Example

Try the following code to implement Bitwise operator in JavaScript.

Live Demo

```
<html>
<body>
  <script type = "text/javascript">
    <!--
      var a = 2; // Bit presentation 10
      var b = 3; // Bit presentation 11
      var linebreak = "<br />";

      document.write("(a & b) => ");
      result = (a & b);
      document.write(result);
      document.write(linebreak);

      document.write("(a | b) => ");
      result = (a | b);
      document.write(result);
      document.write(linebreak);

      document.write("(a ^ b) => ");
      result = (a ^ b);
      document.write(result);
      document.write(linebreak);

      document.write("(~b) => ");
      result = (~b);
      document.write(result);
      document.write(linebreak);

      document.write("(a << b) => ");
      result = (a << b);
      document.write(result);
      document.write(linebreak);

      document.write("(a >> b) => ");
      result = (a >> b);
      document.write(result);
      document.write(linebreak);
    //-->
  </script>
  <p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

(a & b) => 2

(a | b) => 3

$(a \wedge b) \Rightarrow 1$

$(\sim b) \Rightarrow -4$

$(a \ll b) \Rightarrow 16$

$(a \gg b) \Rightarrow 0$

Set the variables to different values and different operators and then try...

Assignment Operators

JavaScript supports the following assignment operators –

| Sr.No. | Operator & Description |
|--------|---|
| 1 | <p>= (Simple Assignment)</p> <p>Assigns values from the right side operand to the left side operand</p> <p>Ex: $C = A + B$ will assign the value of $A + B$ into C</p> |
| 2 | <p>+= (Add and Assignment)</p> <p>It adds the right operand to the left operand and assigns the result to the left operand.</p> <p>Ex: $C += A$ is equivalent to $C = C + A$</p> |
| 3 | <p>-- (Subtract and Assignment)</p> <p>It subtracts the right operand from the left operand and assigns the result to the left operand.</p> <p>Ex: $C -= A$ is equivalent to $C = C - A$</p> |
| 4 | <p>*= (Multiply and Assignment)</p> <p>It multiplies the right operand with the left operand and assigns the result to the left operand.</p> <p>Ex: $C *= A$ is equivalent to $C = C * A$</p> |
| 5 | <p>/= (Divide and Assignment)</p> <p>It divides the left operand with the right operand and assigns the result to the left operand.</p> <p>Ex: $C /= A$ is equivalent to $C = C / A$</p> |
| 6 | <p>%= (Modules and Assignment)</p> <p>It takes modulus using two operands and assigns the result to the left operand.</p> <p>Ex: $C \% = A$ is equivalent to $C = C \% A$</p> |

Note – Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

Example

Try the following code to implement assignment operator in JavaScript.

Live Demo

```
<html>
<body>
  <script type = "text/javascript">
    <!--
      var a = 33;
      var b = 10;
      var linebreak = "<br />";

      document.write("Value of a => (a = b) => ");
      result = (a = b);
      document.write(result);
      document.write(linebreak);

      document.write("Value of a => (a += b) => ");
      result = (a += b);
      document.write(result);
      document.write(linebreak);

      document.write("Value of a => (a -= b) => ");
      result = (a -= b);
      document.write(result);
      document.write(linebreak);

      document.write("Value of a => (a *= b) => ");
      result = (a *= b);
      document.write(result);
      document.write(linebreak);

      document.write("Value of a => (a /= b) => ");
      result = (a /= b);
      document.write(result);
      document.write(linebreak);

      document.write("Value of a => (a %= b) => ");
      result = (a %= b);
      document.write(result);
      document.write(linebreak);
    //-->
  </script>
  <p>Set the variables to different values and different operators and then try...</p>
</body>
```

</html>

Output

Value of a => (a = b) => 10

Value of a => (a += b) => 20

Value of a => (a -= b) => 10

Value of a => (a *= b) => 100

Value of a => (a /= b) => 10

Value of a => (a %= b) => 0

Set the variables to different values and different operators and then try...

Miscellaneous Operator

We will discuss two operators here that are quite useful in JavaScript: the **conditional operator** (? :) and the **typeof operator**.

Conditional Operator (? :)

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

| Sr.No. | Operator and Description |
|--------|--|
| 1 | ? : (Conditional) If Condition is true? Then value X : Otherwise value Y |

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

[Live Demo](#)

```
<html>
<body>
  <script type = "text/javascript">
    <!--
      var a = 10;
      var b = 20;
      var linebreak = "<br />";

      document.write ("((a > b) ? 100 : 200) => ");
      result = (a > b) ? 100 : 200;
      document.write(result);
      document.write(linebreak);

      document.write ("((a < b) ? 100 : 200) => ");
      result = (a < b) ? 100 : 200;
```

```
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

Output

((a > b) ? 100 : 200) => 200

((a < b) ? 100 : 200) => 100

Set the variables to different values and different operators and then try...

typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

| Type | String Returned by typeof |
|-----------|---------------------------|
| Number | "number" |
| String | "string" |
| Boolean | "boolean" |
| Object | "object" |
| Function | "function" |
| Undefined | "undefined" |
| Null | "object" |

Example

The following code shows how to implement **typeof** operator.

[Live Demo](#)

```
<html>
<body>
  <script type = "text/javascript">
    <!--
      var a = 10;
      var b = "String";
      var linebreak = "<br />";

      result = (typeof b == "string" ? "B is String" : "B is Numeric");
      document.write("Result => ");
      document.write(result);
      document.write(linebreak);

      result = (typeof a == "string" ? "A is String" : "A is Numeric");
      document.write("Result => ");
      document.write(result);
      document.write(linebreak);
    //-->
  </script>
  <p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

Output

Result => B is String

Result => A is Numeric

Set the variables to different values and different operators and then try...

JavaScript - if...else Statement

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the **if..else** statement.

Flow Chart of if-else

The following flow chart shows how the if-else statement works.

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

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

if statement

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

Syntax

The syntax for a basic if statement is as follows –

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

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

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

[Live Demo](#)

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

Output

Qualifies for driving

Set the variable to different value and then try...

if...else statement

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

Syntax

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

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

[Live Demo](#)

```
<html>  
<body>  
  <script type = "text/javascript">  
    <!--  
      var age = 15;  
  
      if( age > 18 ) {  
        document.write("<b>Qualifies for driving</b>");  
      } else {  
        document.write("<b>Does not qualify for driving</b>");  
      }  
    //-->  
  </script>  
  <p>Set the variable to different value and then try...</p>  
</body>  
</html>
```

Output

Does not qualify for driving

Set the variable to different value and then try...

if...else if... statement

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

Syntax

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

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

There is nothing special about this code. It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

[Live Demo](#)

```
<html>  
<body>  
  <script type = "text/javascript">  
    <!--  
      var book = "maths";  
      if( book == "history" ) {  
        document.write("<b>History Book</b>");  
      } else if( book == "maths" ) {  
        document.write("<b>Maths Book</b>");  
      } else if( book == "economics" ) {  
        document.write("<b>Economics Book</b>");  
      } else {  
        document.write("<b>Unknown Book</b>");  
      }  
    //-->  
  </script>  
  <p>Set the variable to different value and then try...</p>  
</body>  
</html>
```

Output

Maths Book

Set the variable to different value and then try...

JavaScript - Switch Case

You can use multiple **if...else...if** statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated **if...else if** statements.

Flow Chart

The following flow chart explains a switch-case statement works.

Syntax

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

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

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

We will explain **break** statement in **Loop Control** chapter.

Example

Try the following example to implement switch-case statement.

[Live Demo](#)

```
<html>  
<body>  
  <script type = "text/javascript">  
    <!--  
    var grade = 'A';
```

```

document.write("Entering switch block<br />");
switch (grade) {
  case 'A': document.write("Good job<br />");
  break;

  case 'B': document.write("Pretty good<br />");
  break;

  case 'C': document.write("Passed<br />");
  break;

  case 'D': document.write("Not so good<br />");
  break;

  case 'F': document.write("Failed<br />");
  break;

  default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
<!-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>

```

Output

Entering switch block

Good job

Exiting switch block

Set the variable to different value and then try...

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

[Live Demo](#)

```

<html>
<body>
  <script type = "text/javascript">
    <!--
      var grade = 'A';
      document.write("Entering switch block<br />");
      switch (grade) {
        case 'A': document.write("Good job<br />");
        case 'B': document.write("Pretty good<br />");
        case 'C': document.write("Passed<br />");
        case 'D': document.write("Not so good<br />");
        case 'F': document.write("Failed<br />");

```

```
        default: document.write("Unknown grade<br />")
    }
    document.write("Exiting switch block");
    //-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
Set the variable to different value and then try...

JavaScript - While Loops

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

JavaScript supports all the necessary loops to ease down the pressure of programming.

The while Loop

The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false**, the loop terminates.

Flow Chart

The flow chart of **while loop** looks as follows –

Syntax

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

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

Example

Try the following example to implement while loop.

```
<html>
<body>

<script type = "text/javascript">
  <!--
  var count = 0;
  document.write("Starting Loop ");

  while (count < 10) {
    document.write("Current Count : " + count + "<br />");
    count++;
  }

  document.write("Loop stopped!");
  //-->
</script>

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

Output

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!

Set the variable to different value and then try...

The 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**.

Flow Chart

The flow chart of a **do-while** loop would be as follows –

Syntax

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

```
do {  
    Statement(s) to be executed;  
} while (expression);
```

Note – Don't miss the semicolon used at the end of the **do...while** loop.

Example

Try the following example to learn how to implement a **do-while** loop in JavaScript.

[Live Demo](#)

```
<html>  
<body>  
  <script type = "text/javascript">  
    <!--  
      var count = 0;  
  
      document.write("Starting Loop" + "<br />");  
      do {  
        document.write("Current Count : " + count + "<br />");  
        count++;  
      }  
  
      while (count < 5);  
      document.write ("Loop stopped!");  
    //-->  
  </script>  
  <p>Set the variable to different value and then try...</p>  
</body>  
</html>
```

Output

```
Starting Loop  
Current Count : 0  
Current Count : 1  
Current Count : 2  
Current Count : 3  
Current Count : 4  
Loop Stopped!  
Set the variable to different value and then try...
```

JavaScript - For Loop

The **for** loop is the most compact form of looping. It 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 a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Flow Chart

The flow chart of a **for** loop in JavaScript would be as follows –

Syntax

The syntax of **for** loop in JavaScript is as follows –

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

Example

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

[Live Demo](#)

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

Output

```
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!

Set the variable to different value and then try...

JavaScript *for...in* loop

The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax

The syntax of 'for..in' loop is –

```
for (variablename in object) {  
    statement or block to execute  
}
```

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

Example

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

[Live Demo](#)

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

Output

Navigator Object Properties

serviceWorker

webkitPersistentStorage

webkitTemporaryStorage

geolocation

doNotTrack

onLine

languages

language

userAgent

product

platform

appVersion

appName

appCodeName

hardwareConcurrency

maxTouchPoints

vendorSub

vendor

productSub

cookieEnabled

mimeTypes

plugins

javaEnabled

getStorageUpdates

getGamepads

webkitGetUserMedia

vibrate

getBattery

sendBeacon

registerProtocolHandler

unregisterProtocolHandler

Exiting from the loop!

Set the variable to different object and then try...

JavaScript - Loop Control

JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Chart

The flow chart of a break statement would look as follows –

Example

The following example illustrates the use of a **break** statement with a while loop. Notice how the loop breaks out early once **x** reaches 5 and reaches to **document.write (..)** statement just below to the closing curly brace –

[Live Demo](#)

```
<html>
<body>
  <script type = "text/javascript">
    <!--
    var x = 1;
    document.write("Entering the loop<br /> ");

    while (x < 20) {
      if (x == 5) {
        break; // breaks out of loop completely
      }
      x = x + 1;
      document.write( x + "<br />");
    }
    document.write("Exiting the loop!<br /> ");
    //-->
  </script>

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

Output

Entering the loop

2

3

4

5

Exiting the loop!

Set the variable to different value and then try...

We already have seen the usage of **break** statement inside a **switch** statement.

The continue Statement

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a **continue** statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example

This example illustrates the use of a **continue** statement with a while loop. Notice how the **continue** statement is used to skip printing when the index held in variable **x** reaches 5 –

[Live Demo](#)

```
<html>
<body>
  <script type = "text/javascript">
    <!--
      var x = 1;
      document.write("Entering the loop<br /> ");

      while (x < 10) {
        x = x + 1;

        if (x == 5) {
          continue; // skip rest of the loop body
        }
        document.write( x + "<br />");
      }
      document.write("Exiting the loop!<br /> ");
    //-->
  </script>
  <p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

Entering the loop

2

3

4

6

7

8

9

10

Exiting the loop!

Set the variable to different value and then try...

Using Labels to Control the Flow

Starting from JavaScript 1.2, a label can be used with **break** and **continue** to control the flow more precisely. A **label** is simply an identifier followed by a colon (:) that is applied to a statement or a block of code. We will see two different examples to understand how to use labels with break and continue.

Note – Line breaks are not allowed between the ‘**continue**’ or ‘**break**’ statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Try the following two examples for a better understanding of Labels.

Example 1

The following example shows how to implement Label with a break statement.

[Live Demo](#)

```
<html>
<body>
  <script type = "text/javascript">
    <!--
      document.write("Entering the loop!<br /> ");
      outerloop:    // This is the label name
      for (var i = 0; i < 5; i++) {
        document.write("Outerloop: " + i + "<br />");
        innerloop:
        for (var j = 0; j < 5; j++) {
          if (j > 3 ) break ;      // Quit the innermost loop
          if (i == 2) break innerloop; // Do the same thing
          if (i == 4) break outerloop; // Quit the outer loop
          document.write("Innerloop: " + j + " <br />");
        }
      }
      document.write("Exiting the loop!<br /> ");
    //-->
  </script>
</body>
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
```

Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
Exiting the loop!

Example 2

Live Demo

```
<html>
<body>

<script type = "text/javascript">
  <!--
  document.write("Entering the loop!<br /> ");
  outerloop: // This is the label name

  for (var i = 0; i < 3; i++) {
    document.write("Outerloop: " + i + "<br />");
    for (var j = 0; j < 5; j++) {
      if (j == 3) {
        continue outerloop;
      }
      document.write("Innerloop: " + j + "<br />");
    }
  }

  document.write("Exiting the loop!<br /> ");
  //-->
</script>

</body>
</html>
```

Output

Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 1
Innerloop: 0
Innerloop: 1

Innerloop: 2
Outerloop: 2
Innerloop: 0
Innerloop: 1
Innerloop: 2
Exiting the loop!

JavaScript - Functions

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

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like **alert()** and **write()** in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

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

Example

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

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

Calling a Function

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

[Live Demo](#)

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

Output

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

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

[Live Demo](#)

```
<html>
<head>
  <script type = "text/javascript">
    function sayHello(name, age) {
      document.write (name + " is " + age + " years old.");
    }
  </script>
```

```
</head>

<body>
  <p>Click the following button to call the function</p>
  <form>
    <input type = "button" onclick = "sayHello('Zara', 7)" value = "Say Hello">
  </form>
  <p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

The return Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

[Live Demo](#)

```
<html>
<head>
  <script type = "text/javascript">
    function concatenate(first, last) {
      var full;
      full = first + last;
      return full;
    }
    function secondFunction() {
      var result;
      result = concatenate("Zara", 'Ali');
      document.write (result );
    }
  </script>
</head>

<body>
  <p>Click the following button to call the function</p>
  <form>
    <input type = "button" onclick = "secondFunction()" value = "Call Function">
  </form>
  <p>Use different parameters inside the function and then try...</p>
```

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

Output

There is a lot to learn about JavaScript functions, however we have covered the most important concepts in this tutorial.

- [JavaScript Nested Functions](#)
- [JavaScript Function\(\) Constructor](#)
- [JavaScript Function Literals](#)

JavaScript - Events

What is an Event ?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Please go through this small tutorial for a better understanding [HTML Event Reference](#). Here we will see a few examples to understand a relation between Event and JavaScript –

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

Try the following example.

[Live Demo](#)

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

```
//-->
</script>
</head>

<body>
  <p>Click the following button and see result</p>
  <form>
    <input type = "button" onclick = "sayHello()" value = "Say Hello" />
  </form>
</body>
</html>
```

Output

onsubmit Event Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use **onsubmit**. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

```
<html>
<head>
  <script type = "text/javascript">
    <!--
      function validation() {
        all validation goes here
        .....
        return either true or false
      }
    <!-->
  </script>
</head>

<body>
  <form method = "POST" action = "t.cgi" onsubmit = "return validate()">
    .....
    <input type = "submit" value = "Submit" />
  </form>
</body>
</html>
```

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

[Live Demo](#)

```
<html>
<head>
  <script type = "text/javascript">
    <!--
      function over() {
        document.write ("Mouse Over");
      }
      function out() {
        document.write ("Mouse Out");
      }
    //-->
  </script>
</head>

<body>
  <p>Bring your mouse inside the division to see the result:</p>
  <div onmouseover = "over()" onmouseout = "out()">
    <h2> This is inside the division </h2>
  </div>
</body>
</html>
```

Output

HTML 5 Standard Events

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

| Attribute | Value | Description |
|-----------|--------|---|
| Offline | script | Triggers when the document goes offline |
| Onabort | script | Triggers on an abort event |

| | | |
|------------------|--------|--|
| onafterprint | script | Triggers after the document is printed |
| onbeforeunload | script | Triggers before the document loads |
| onbeforeprint | script | Triggers before the document is printed |
| onblur | script | Triggers when the window loses focus |
| oncanplay | script | Triggers when media can start play, but might has to stop for buffering |
| oncanplaythrough | script | Triggers when media can be played to the end, without stopping for buffering |
| onchange | script | Triggers when an element changes |
| onclick | script | Triggers on a mouse click |
| oncontextmenu | script | Triggers when a context menu is triggered |
| ondblclick | script | Triggers on a mouse double-click |
| ondrag | script | Triggers when an element is dragged |
| ondragend | script | Triggers at the end of a drag operation |
| ondragenter | script | Triggers when an element has been dragged to a valid drop target |
| ondragleave | script | Triggers when an element is being dragged over a valid drop target |
| ondragover | script | Triggers at the start of a drag operation |
| ondragstart | script | Triggers at the start of a drag operation |

| | | |
|------------------|--------|--|
| ondrop | script | Triggers when dragged element is being dropped |
| ondurationchange | script | Triggers when the length of the media is changed |
| onemptied | script | Triggers when a media resource element suddenly becomes empty. |
| onended | script | Triggers when media has reach the end |
| onerror | script | Triggers when an error occur |
| onfocus | script | Triggers when the window gets focus |
| onformchange | script | Triggers when a form changes |
| onforminput | script | Triggers when a form gets user input |
| onhaschange | script | Triggers when the document has change |
| oninput | script | Triggers when an element gets user input |
| oninvalid | script | Triggers when an element is invalid |
| onkeydown | script | Triggers when a key is pressed |
| onkeypress | script | Triggers when a key is pressed and released |
| onkeyup | script | Triggers when a key is released |
| onload | script | Triggers when the document loads |
| onloadeddata | script | Triggers when media data is loaded |

| | | |
|------------------|--------|--|
| onloadedmetadata | script | Triggers when the duration and other media data of a media element is loaded |
| onloadstart | script | Triggers when the browser starts to load the media data |
| onmessage | script | Triggers when the message is triggered |
| onmousedown | script | Triggers when a mouse button is pressed |
| onmousemove | script | Triggers when the mouse pointer moves |
| onmouseout | script | Triggers when the mouse pointer moves out of an element |
| onmouseover | script | Triggers when the mouse pointer moves over an element |
| onmouseup | script | Triggers when a mouse button is released |
| onmousewheel | script | Triggers when the mouse wheel is being rotated |
| onoffline | script | Triggers when the document goes offline |
| ononline | script | Triggers when the document comes online |
| ononline | script | Triggers when the document comes online |
| onpagehide | script | Triggers when the window is hidden |
| onpageshow | script | Triggers when the window becomes visible |
| onpause | script | Triggers when media data is paused |
| onplay | script | Triggers when media data is going to start playing |

| | | |
|--------------------|--------|--|
| onplaying | script | Triggers when media data has start playing |
| onpopstate | script | Triggers when the window's history changes |
| onprogress | script | Triggers when the browser is fetching the media data |
| onratechange | script | Triggers when the media data's playing rate has changed |
| onreadystatechange | script | Triggers when the ready-state changes |
| onredo | script | Triggers when the document performs a redo |
| onresize | script | Triggers when the window is resized |
| onscroll | script | Triggers when an element's scrollbar is being scrolled |
| onseeked | script | Triggers when a media element's seeking attribute is no longer true, and the seeking has ended |
| onseeking | script | Triggers when a media element's seeking attribute is true, and the seeking has begun |
| onselect | script | Triggers when an element is selected |
| onstalled | script | Triggers when there is an error in fetching media data |
| onstorage | script | Triggers when a document loads |
| onsubmit | script | Triggers when a form is submitted |
| onsuspend | script | Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched |

| | | |
|----------------|--------|---|
| onupdate | script | Triggers when media changes its playing position |
| onundo | script | Triggers when a document performs an undo |
| onunload | script | Triggers when the user leaves the document |
| onvolumechange | script | Triggers when media changes the volume, also when volume is set to "mute" |
| onwaiting | script | Triggers when media has stopped playing, but is expected to resume |

JavaScript and Cookies

What are Cookies ?

Web Browsers and Servers use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website, it is required to maintain session information among different pages. For example, one user registration ends after completing many pages. But how to maintain users' session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

How It Works ?

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields –

- **Expires** – The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain** – The domain name of your site.
- **Path** – The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure** – If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value** – Cookies are set and retrieved in the form of key-value pairs

Cookies were originally designed for CGI programming. The data contained in a cookie is automatically transmitted between the web browser and the web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the **cookie** property of the **Document** object. JavaScript can read, create, modify, and delete the cookies that apply to the current web page.

JavaScript - Objects

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers –

- **Encapsulation** – the capability to store related information, whether data or methods, together in an object.
- **Aggregation** – the capability to store one object inside another object.
- **Inheritance** – the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** – the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is –

```
objectName.objectProperty = propertyValue;
```

For example – The following code gets the document title using the **"title"** property of the **document** object.

```
var str = document.title;
```

Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

For example – Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

```
document.write("This is test");
```

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are `Object()`, `Array()`, and `Date()`. These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

The Object() Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

Example 1

Try the following example; it demonstrates how to create an Object.

[Live Demo](#)

```
<html>
  <head>
    <title>User-defined objects</title>
    <script type = "text/javascript">
      var book = new Object(); // Create the object
      book.subject = "Perl"; // Assign properties to the object
      book.author = "Mohtashim";
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      document.write("Book name is : " + book.subject + "<br>");
      document.write("Book author is : " + book.author + "<br>");
    </script>
  </body>
</html>
```

Output

Book name is : Perl
Book author is : Mohtashim

Example 2

This example demonstrates how to create an object with a User-Defined Function. Here **this** keyword is used to refer to the object that has been passed to a function.

[Live Demo](#)

```
<html>
  <head>
    <title>User-defined objects</title>
    <script type = "text/javascript">
      function book(title, author) {
        this.title = title;
        this.author = author;
      }
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      var myBook = new book("Perl", "Mohtashim");
      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
    </script>
  </body>
</html>
```

Output

Book title is : Perl
Book author is : Mohtashim
Defining Methods for an Object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

Example

Try the following example; it shows how to add a function along with an object.

[Live Demo](#)

```
<html>

  <head>
```

```

<title>User-defined objects</title>
<script type = "text/javascript">
  // Define a function which will work as a method
  function addPrice(amount) {
    this.price = amount;
  }

  function book(title, author) {
    this.title = title;
    this.author = author;
    this.addPrice = addPrice; // Assign that method as property.
  }
</script>
</head>

<body>
<script type = "text/javascript">
  var myBook = new book("Perl", "Mohtashim");
  myBook.addPrice(100);

  document.write("Book title is : " + myBook.title + "<br>");
  document.write("Book author is : " + myBook.author + "<br>");
  document.write("Book price is : " + myBook.price + "<br>");
</script>
</body>
</html>

```

Output

```

Book title is : Perl
Book author is : Mohtashim
Book price is : 100
The 'with' Keyword

```

The **'with'** keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to **with** becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

Syntax

The syntax for with object is as follows –

```

with (object) {
  properties used without the object name and dot
}

```

Example

Try the following example.

```
<html>
  <head>
    <title>User-defined objects</title>
    <script type = "text/javascript">
      // Define a function which will work as a method
      function addPrice(amount) {
        with(this) {
          price = amount;
        }
      }
      function book(title, author) {
        this.title = title;
        this.author = author;
        this.price = 0;
        this.addPrice = addPrice; // Assign that method as property.
      }
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      var myBook = new book("Perl", "Mohtashim");
      myBook.addPrice(100);

      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
      document.write("Book price is : " + myBook.price + "<br>");
    </script>
  </body>
</html>
```

Output

Book title is : Perl

Book author is : Mohtashim

Book price is : 100

JavaScript Native Objects

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects –

- [JavaScript Number Object](#)
- [JavaScript Boolean Object](#)
- [JavaScript String Object](#)

- [JavaScript Array Object](#)
- [JavaScript Date Object](#)
- [JavaScript Math Object](#)
- [JavaScript RegExp Object](#)

JavaScript - The Number Object

The **Number** object represents numerical data, either integers or floating-point numbers. In general, you do not need to worry about **Number** objects because the browser automatically converts number literals to instances of the number class.

Syntax

The syntax for creating a **number** object is as follows –

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

Number Properties

Here is a list of each property and their description.

| Sr.No. | Property & Description |
|--------|--|
| 1 | <u>MAX_VALUE</u> The largest possible value a number in JavaScript can have 1.7976931348623157E+308 |
| 2 | <u>MIN_VALUE</u> The smallest possible value a number in JavaScript can have 5E-324 |
| 3 | <u>NaN</u> Equal to a value that is not a number. |
| 4 | <u>NEGATIVE_INFINITY</u> A value that is less than MIN_VALUE. |
| 5 | <u>POSITIVE_INFINITY</u> A value that is greater than MAX_VALUE |

| | |
|---|--|
| 6 | <u>prototype</u> A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document |
| 7 | <u>constructor</u> Returns the function that created this object's instance. By default this is the Number object. |

In the following sections, we will take a few examples to demonstrate the properties of Number.

Number Methods

The Number object contains only the default methods that are a part of every object's definition.

| Sr.No. | Method & Description |
|--------|---|
| 1 | <u>toExponential()</u> Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation. |
| 2 | <u>toFixed()</u> Formats a number with a specific number of digits to the right of the decimal. |
| 3 | <u>toLocaleString()</u> Returns a string value version of the current number in a format that may vary according to a browser's local settings. |
| 4 | <u>toPrecision()</u> Defines how many total digits (including digits to the left and right of the decimal) to display of a number. |
| 5 | <u>toString()</u> Returns the string representation of the number's value. |
| 6 | <u>valueOf()</u> Returns the number's value. |

In the following sections, we will have a few examples to explain the methods of Number.

JavaScript - The Boolean Object

The **Boolean** object represents two values, either "true" or "false". If *value* parameter is omitted or is 0, -0, null, false, **NaN**, undefined, or the empty string (""), the object has an initial value of false.

Syntax

Use the following syntax to create a **boolean** object.

```
var val = new Boolean(value);
```

Boolean Properties

Here is a list of the properties of Boolean object –

| Sr.No. | Property & Description |
|--------|---|
| 1 | <u>constructor</u> Returns a reference to the Boolean function that created the object. |
| 2 | <u>prototype</u> The prototype property allows you to add properties and methods to an object. |

In the following sections, we will have a few examples to illustrate the properties of Boolean object.

Boolean Methods

Here is a list of the methods of Boolean object and their description.

| Sr.No. | Method & Description |
|--------|--|
| 1 | <u>toSource()</u> Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object. |
| 2 | <u>toString()</u> Returns a string of either "true" or "false" depending upon the value of the object. |
| 3 | <u>valueOf()</u> |

| | |
|--|--|
| | Returns the primitive value of the Boolean object. |
|--|--|

In the following sections, we will have a few examples to demonstrate the usage of the Boolean methods.

JavaScript - The Strings Object

The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

Syntax

Use the following syntax to create a String object –

```
var val = new String(string);
```

The **String** parameter is a series of characters that has been properly encoded.

String Properties

Here is a list of the properties of String object and their description.

| Sr.No. | Property & Description |
|--------|---|
| 1 | <u>constructor</u> Returns a reference to the String function that created the object. |
| 2 | <u>length</u> Returns the length of the string. |
| 3 | <u>prototype</u> The prototype property allows you to add properties and methods to an object. |

In the following sections, we will have a few examples to demonstrate the usage of String properties.

String Methods

Here is a list of the methods available in String object along with their description.

| Sr.No. | Method & Description |
|--------|----------------------|
|--------|----------------------|

| | |
|----|--|
| 1 | <u>charAt()</u> Returns the character at the specified index. |
| 2 | <u>charCodeAt()</u> Returns a number indicating the Unicode value of the character at the given index. |
| 3 | <u>concat()</u> Combines the text of two strings and returns a new string. |
| 4 | <u>indexOf()</u> Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found. |
| 5 | <u>lastIndexOf()</u> Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found. |
| 6 | <u>localeCompare()</u> Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order. |
| 7 | <u>match()</u> Used to match a regular expression against a string. |
| 8 | <u>replace()</u> Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring. |
| 9 | <u>search()</u> Executes the search for a match between a regular expression and a specified string. |
| 10 | <u>slice()</u> Extracts a section of a string and returns a new string. |

| | |
|----|---|
| 11 | <u>split()</u> Splits a String object into an array of strings by separating the string into substrings. |
| 12 | <u>substr()</u> Returns the characters in a string beginning at the specified location through the specified number of characters. |
| 13 | <u>substring()</u> Returns the characters in a string between two indexes into the string. |
| 14 | <u>toLocaleLowerCase()</u> The characters within a string are converted to lower case while respecting the current locale. |
| 15 | <u>toLocaleUpperCase()</u> The characters within a string are converted to upper case while respecting the current locale. |
| 16 | <u>toLowerCase()</u> Returns the calling string value converted to lower case. |
| 17 | <u>toString()</u> Returns a string representing the specified object. |
| 18 | <u>toUpperCase()</u> Returns the calling string value converted to uppercase. |
| 19 | <u>valueOf()</u> Returns the primitive value of the specified object. |

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
}
```

JavaScript - Errors & Exceptions Handling

There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

Syntax Errors

Syntax errors, also called **parsing errors**, occur at compile time in traditional programming languages and at interpret time in JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script type = "text/javascript">
  <!--
    window.print(
  //-->
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

Runtime Errors

Runtime errors, also called **exceptions**, occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script type = "text/javascript">
  <!--
    window.printme();
  //-->
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions.

You can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

Here is the **try...catch...finally** block syntax –

```
<script type = "text/javascript">
  <!--
    try {
      // Code to run
      [break;]
    }

    catch ( e ) {
      // Code to run if an exception occurs
      [break;]
    }

    [ finally {
      // Code that is always executed regardless of
      // an exception occurring
    } ]
  //-->
</script>
```

The **try** block must be followed by either exactly one **catch** block or one **finally** block (or one of both). When an exception occurs in the **try** block, the exception is placed in **e** and the **catch** block is executed. The optional **finally** block executes unconditionally after try/catch.

Examples

Here is an example where we are trying to call a non-existing function which in turn is raising an exception. Let us see how it behaves without **try...catch**–

[Live Demo](#)

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function myFunc() {
          var a = 100;
          alert("Value of variable a is : " + a );
        }
      //-->
    </script>
  </head>
</html>
```

```
    </script>
</head>

<body>
  <p>Click the following to see the result:</p>

  <form>
    <input type = "button" value = "Click Me" onclick = "myFunc();"
  />
  </form>
</body>
</html>
```

Output

Now let us try to catch this exception using **try...catch** and display a user-friendly message. You can also suppress this message, if you want to hide this error from a user.

[Live Demo](#)

```
<html>
  <head>

    <script type = "text/javascript">
      <!--
        function myFunc() {
          var a = 100;
          try {
            alert("Value of variable a is : " + a );
          }
          catch ( e ) {
            alert("Error: " + e.description );
          }
        }
      //-->
    </script>

  </head>
  <body>
    <p>Click the following to see the result:</p>

    <form>
      <input type = "button" value = "Click Me" onclick = "myFunc();"
    />
    </form>

  </body>
</html>
```

Output

You can use **finally** block which will always execute unconditionally after the try/catch. Here is an example.

[Live Demo](#)

```
<html>
  <head>

    <script type = "text/javascript">
      <!--
        function myFunc() {
          var a = 100;

          try {
            alert("Value of variable a is : " + a );
          }
          catch ( e ) {
            alert("Error: " + e.description );
          }
          finally {
            alert("Finally block will always execute!" );
          }
        }
      //-->
    </script>

  </head>
  <body>
    <p>Click the following to see the result:</p>

    <form>
      <input type = "button" value = "Click Me" onclick = "myFunc();"
/>
    </form>

  </body>
</html>
```

Output

The throw Statement

You can use **throw** statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.

Example

The following example demonstrates how to use a **throw** statement.

```
<html>
  <head>

    <script type = "text/javascript">
      <!--
        function myFunc() {
          var a = 100;
          var b = 0;

          try {
            if ( b == 0 ) {
              throw( "Divide by zero error." );
            } else {
              var c = a / b;
            }
          }
          catch ( e ) {
            alert("Error: " + e );
          }
        }
      //-->
    </script>

  </head>
  <body>
    <p>Click the following to see the result:</p>

    <form>
      <input type = "button" value = "Click Me" onclick = "myFunc();"
    />
    </form>

  </body>
</html>
```

Output

You can raise an exception in one function using a string, integer, Boolean, or an object and then you can capture that exception either in the same function as we did above, or in another function using a **try...catch** block.

The onerror() Method

The **onerror** event handler was the first feature to facilitate error handling in JavaScript. The **error** event is fired on the window object whenever an exception occurs on the page.

```
<html>
  <head>

    <script type = "text/javascript">
      <!--
        window.onerror = function () {
          alert("An error occurred.");
        }
      //-->
    </script>

  </head>
  <body>
    <p>Click the following to see the result:</p>

    <form>
      <input type = "button" value = "Click Me" onclick = "myFunc();"
/>
    </form>

  </body>
</html>
```

Output

The **onerror** event handler provides three pieces of information to identify the exact nature of the error –

- **Error message** – The same message that the browser would display for the given error
- **URL** – The file in which the error occurred
- **Line number**– The line number in the given URL that caused the error

Here is the example to show how to extract this information.

Example

[Live Demo](#)

```
<html>
  <head>

    <script type = "text/javascript">
      <!--
        window.onerror = function (msg, url, line) {
          alert("Message : " + msg );
          alert("url : " + url );
          alert("Line number : " + line );
        }
      //-->
    </script>
```

```
</head>
<body>
  <p>Click the following to see the result:</p>

  <form>
    <input type = "button" value = "Click Me" onclick = "myFunc();"
  />
  </form>

</body>
</html>
```

Output

You can display extracted information in whatever way you think it is better.

You can use an **onerror** method, as shown below, to display an error message in case there is any problem in loading an image.

```

```

You can use **onerror** with many HTML tags to display appropriate messages in case of errors.

JavaScript - Form Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** – First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** – Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Example

We will take an example to understand the process of validation. Here is a simple form in html format.

[Live Demo](#)

```
<html>
  <head>
```

```

<title>Form Validation</title>
<script type = "text/javascript">
  <!--
    // Form validation code will come here.
  //-->
</script>
</head>

<body>
  <form action = "/cgi-bin/test.cgi" name = "myForm" onsubmit =
"return(validate());">
    <table cellspacing = "2" cellpadding = "2" border = "1">

      <tr>
        <td align = "right">Name</td>
        <td><input type = "text" name = "Name" /></td>
      </tr>

      <tr>
        <td align = "right">EMail</td>
        <td><input type = "text" name = "EMail" /></td>
      </tr>

      <tr>
        <td align = "right">Zip Code</td>
        <td><input type = "text" name = "Zip" /></td>
      </tr>

      <tr>
        <td align = "right">Country</td>
        <td>
          <select name = "Country">
            <option value = "-1" selected>[choose
yours]</option>
            <option value = "1">USA</option>
            <option value = "2">UK</option>
            <option value = "3">INDIA</option>
          </select>
        </td>
      </tr>

      <tr>
        <td align = "right"></td>
        <td><input type = "submit" value = "Submit" /></td>
      </tr>

    </table>
  </form>
</body>
</html>

```

Output

Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

```
<script type = "text/javascript">
  <!--
    // Form validation code will come here.
    function validate() {

      if( document.myForm.Name.value == "" ) {
        alert( "Please provide your name!" );
        document.myForm.Name.focus() ;
        return false;
      }
      if( document.myForm.EMail.value == "" ) {
        alert( "Please provide your Email!" );
        document.myForm.EMail.focus() ;
        return false;
      }
      if( document.myForm.Zip.value == "" || isNaN(
document.myForm.Zip.value ) ||
        document.myForm.Zip.value.length != 5 ) {

        alert( "Please provide a zip in the format #####." );
        document.myForm.Zip.focus() ;
        return false;
      }
      if( document.myForm.Country.value == "-1" ) {
        alert( "Please provide your country!" );
        return false;
      }
      return( true );
    }
  //-->
</script>
```

Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Example

Try the following code for email validation.

```
<script type = "text/javascript">
  <!--
    function validateEmail() {
      var emailID = document.myForm.Email.value;
      atpos = emailID.indexOf("@");
      dotpos = emailID.lastIndexOf(".");

      if (atpos < 1 || ( dotpos - atpos < 2 )) {
        alert("Please enter correct email ID")
        document.myForm.Email.focus() ;
        return false;
      }
      return( true );
    }
  //-->
</script>
```

Unit – IV

JSP

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

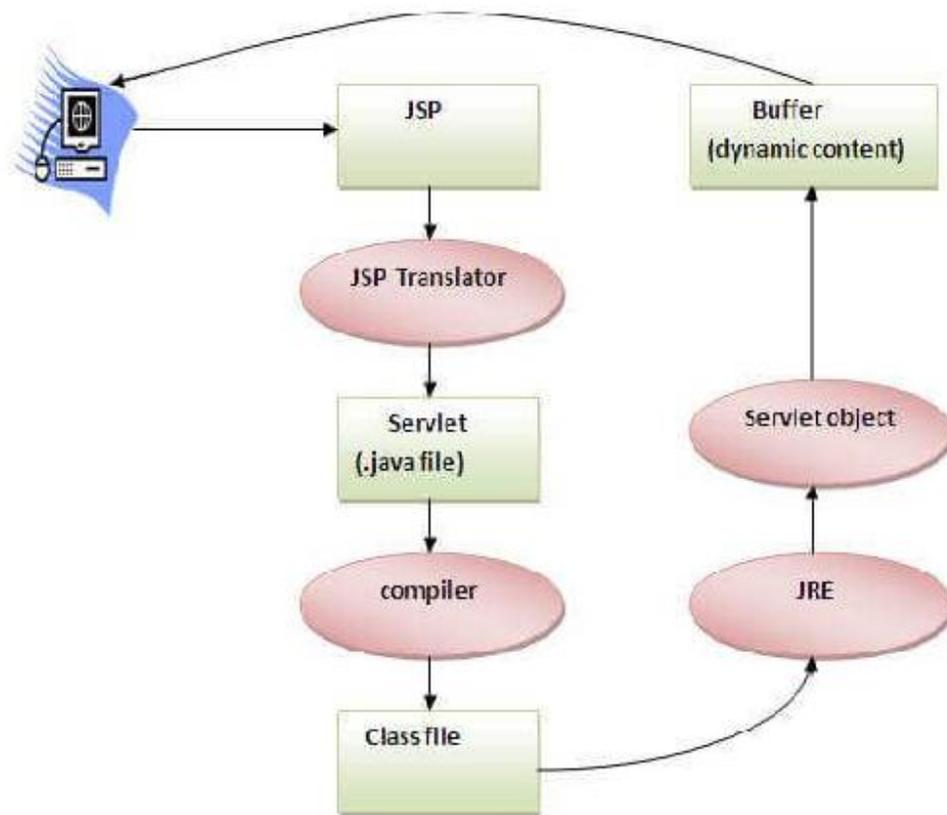
4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes `jspInit()` method).
- Request processing (the container invokes `_jspService()` method).
- Destroy (the container invokes `jspDestroy()` method).



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

Creating a simple JSP Page

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

1. <html>
2. <body>
3. <% out.print(2*5); %>

- 4. </body>
- 5. </html>

It will print **10** on the browser.

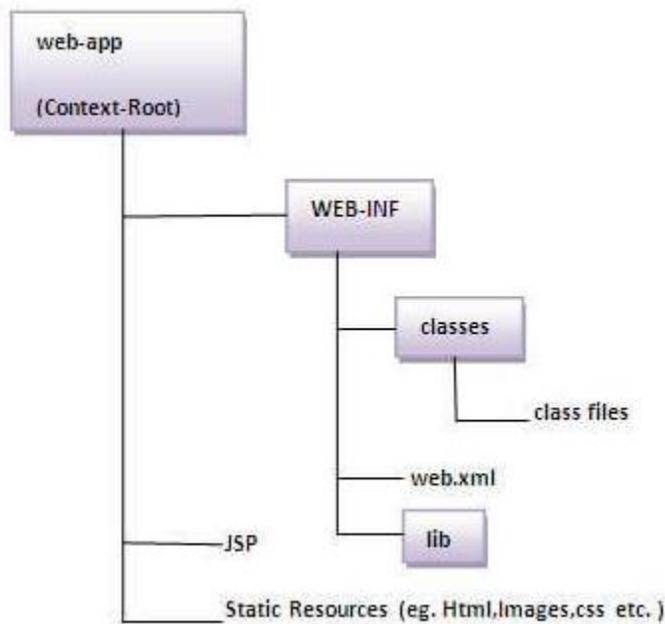
How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

- o Start the server
- o Put the JSP file in a folder and deploy on the server
- o Visit the browser by the URL <http://localhost:portno/contextRoot/jspfile>, for example, <http://localhost:8888/myapplication/index.jsp>

The Directory structure of JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



Translation and Compilation

During the translation phase each type of data in a JSP page is treated differently. Static data is transformed into code that will emit the data into the response stream. JSP elements are treated as follows:

- Directives are used to control how the web container translates and executes the JSP page.
- Scripting elements are inserted into the JSP page's servlet class.
- Expression language expressions are passed as parameters to calls to the JSP expression evaluator.
- `jsp:[set|get]Property` elements are converted into method calls to JavaBeans components.
- `jsp:[include|forward]` elements are converted into invocations of the Java Servlet API.
- The `jsp:plugin` element is converted into browser-specific markup for activating an applet.
- Custom tags are converted into calls to the tag handler that implements the custom tag.

In the Application Server, the source for the servlet created from a JSP page named *pageName* is in this file:

```
domain-dir/generated/jsp/j2ee-modules/WAR-NAME/pageName_jsp.java
```

For example, the source for the index page (named `index.jsp`) for the date localization example discussed at the beginning of the chapter would be named:

```
domain-dir/generated/jsp/j2ee-modules/date/index_jsp.java
```

Both the translation and the compilation phases can yield errors that are observed only when the page is requested for the first time. If an error is encountered during either phase, the server will return `JasperException` and a message that includes the name of the JSP page and the line where the error occurred.

After the page has been translated and compiled, the JSP page's servlet (for the most part) follows the servlet life cycle described in *Servlet Life Cycle*:

1. If an instance of the JSP page's servlet does not exist, the container:

- A. Loads the JSP page's servlet class
 - B. Instantiates an instance of the servlet class
 - C. Initializes the servlet instance by calling the `jspInit` method
2. The container invokes the `_jspService` method, passing request and response objects.

If the container needs to remove the JSP page's servlet, it calls the `jspDestroy` method.

- The Example JSP Pages
- Execution

Directive

A JSP directive affects the overall structure of the servlet class. It usually has the following form –

```
<%@ directive attribute = "value" %>
```

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.

The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

There are three types of directive tag –

| S.No. | Directive & Description |
|-------|--|
| 1 | <pre><%@ page ... %></pre> <p>Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.</p> |
| 2 | <pre><%@ include ... %></pre> <p>Includes a file during the translation phase.</p> |
| 3 | <pre><%@ taglib ... %></pre> <p>Declares a tag library, containing custom actions, used in the page</p> |

JSP - The page Directive

The **page** directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of the page directive –

<%@ page attribute = "value" %>

You can write the XML equivalent of the above syntax as follows –

<jsp:directive.page attribute = "value" />

Attributes

Following table lists out the attributes associated with the page directive –

| S.No. | Attribute & Purpose |
|-------|---|
| 1 | buffer Specifies a buffering model for the output stream. |
| 2 | autoFlush Controls the behavior of the servlet output buffer. |
| 3 | contentType Defines the character encoding scheme. |
| 4 | errorPage Defines the URL of another JSP that reports on Java unchecked runtime exceptions. |
| 5 | isErrorPage Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute. |
| 6 | extends Specifies a superclass that the generated servlet must extend. |
| 7 | import Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes. |
| 8 | info Defines a string that can be accessed with the servlet's getServletInfo() method. |

| | |
|----|---|
| 9 | isThreadSafe Defines the threading model for the generated servlet. |
| 10 | language Defines the programming language used in the JSP page. |
| 11 | session Specifies whether or not the JSP page participates in HTTP sessions |
| 12 | isELIgnored Specifies whether or not the EL expression within the JSP page will be ignored. |
| 13 | isScriptingEnabled Determines if the scripting elements are allowed for use. |

Check for more details related to all the above attributes at [Page Directive](#).

The include Directive

The **include** directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the **include** directives anywhere in your JSP page.

The general usage form of this directive is as follows –

```
<%@ include file = "relative url" >
```

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write the XML equivalent of the above syntax as follows –

```
<jsp:directive.include file = "relative url" />
```

For more details related to include directive, check the [Include Directive](#).

The taglib Directive

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

The taglib directive follows the syntax given below –

```
<%@ taglib uri="uri" prefix = "prefixOfTag" >
```

Here, the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

You can write the XML equivalent of the above syntax as follows –

```
<jsp:directive.taglib uri = "uri" prefix = "prefixOfTag" />
```

For more details related to the taglib directive, check the [Taglib Directive](#).

the Implicit Objects in JSP. These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared. JSP Implicit Objects are also called **pre-defined variables**.

Following table lists out the nine Implicit Objects that JSP supports –

| S.No. | Object & Description |
|-------|---|
| 1 | request This is the HttpServletRequest object associated with the request. |
| 2 | response This is the HttpServletResponse object associated with the response to the client. |
| 3 | out This is the PrintWriter object used to send output to the client. |
| 4 | session This is the HttpSession object associated with the request. |
| 5 | application This is the ServletContext object associated with the application context. |
| 6 | config This is the ServletConfig object associated with the page. |
| 7 | pageContext This encapsulates use of server-specific features like higher performance JspWriters . |

| | |
|---|--|
| 8 | <p>page</p> <p>This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.</p> |
| 9 | <p>Exception</p> <p>The Exception object allows the exception data to be accessed by designated JSP.</p> |

The request Object

The request object is an instance of a **javax.servlet.http.HttpServletRequest** object. Each time a client requests a page the JSP engine creates a new object to represent that request.

The request object provides methods to get the HTTP header information including form data, cookies, HTTP methods etc.

We can cover a complete set of methods associated with the request object in a subsequent chapter – [JSP - Client Request](#).

The response Object

The response object is an instance of a **javax.servlet.http.HttpServletResponse** object. Just as the server creates the request object, it also creates an object to represent the response to the client.

The response object also defines the interfaces that deal with creating new HTTP headers. Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes, etc.

We will cover a complete set of methods associated with the response object in a subsequent chapter – [JSP - Server Response](#).

The out Object

The out implicit object is an instance of a **javax.servlet.jsp.JspWriter** object and is used to send content in a response.

The initial JspWriter object is instantiated differently depending on whether the page is buffered or not. Buffering can be easily turned off by using the **buffered = 'false'** attribute of the page directive.

The JspWriter object contains most of the same methods as the **java.io.PrintWriter** class. However, JspWriter has some additional methods designed to deal with buffering. Unlike the PrintWriter object, JspWriter throws **IOExceptions**.

Following table lists out the important methods that we will use to write **boolean char, int, double, object, String**, etc.

| S.No. | Method & Description |
|-------|----------------------|
|-------|----------------------|

| | |
|---|---|
| 1 | out.print(dataType dt) Print a data type value |
| 2 | out.println(dataType dt) Print a data type value then terminate the line with new line character. |
| 3 | out.flush() Flush the stream. |

The session Object

The session object is an instance of **javax.servlet.http.HttpSession** and behaves exactly the same way that session objects behave under Java Servlets.

The session object is used to track client session between client requests. We will cover the complete usage of session object in a subsequent chapter – [JSP - Session Tracking](#).

The application Object

The application object is direct wrapper around the **ServletContext** object for the generated Servlet and in reality an instance of a **javax.servlet.ServletContext** object.

This object is a representation of the JSP page through its entire lifecycle. This object is created when the JSP page is initialized and will be removed when the JSP page is removed by the **jspDestroy()** method.

By adding an attribute to application, you can ensure that all JSP files that make up your web application have access to it.

We will check the use of Application Object in [JSP - Hits Counter](#) chapter.

The config Object

The config object is an instantiation of **javax.servlet.ServletConfig** and is a direct wrapper around the **ServletConfig** object for the generated servlet.

This object allows the JSP programmer access to the Servlet or JSP engine initialization parameters such as the paths or file locations etc.

The following **config** method is the only one you might ever use, and its usage is trivial –

```
config.getServletName();
```

This returns the servlet name, which is the string contained in the **<servlet-name>** element defined in the **WEB-INF\web.xml** file.

The pageContext Object

The pageContext object is an instance of a **javax.servlet.jsp.PageContext** object. The pageContext object is used to represent the entire JSP page.

This object is intended as a means to access information about the page while avoiding most of the implementation details.

This object stores references to the request and response objects for each request. The **application**, **config**, **session**, and out objects are derived by accessing attributes of this object.

The pageContext object also contains information about the directives issued to the JSP page, including the buffering information, the errorPageURL, and page scope.

The PageContext class defines several fields, including **PAGE_SCOPE**, **REQUEST_SCOPE**, **SESSION_SCOPE**, and **APPLICATION_SCOPE**, which identify the four scopes. It also supports more than 40 methods, about half of which are inherited from the **javax.servlet.jsp.JspContext** class.

One of the important methods is **removeAttribute**. This method accepts either one or two arguments. For example, **pageContext.removeAttribute ("attrName")** removes the attribute from all scopes, while the following code only removes it from the page scope –

```
pageContext.removeAttribute("attrName", PAGE_SCOPE);
```

The use of pageContext can be checked in [JSP - File Uploading](#) chapter.

The page Object

This object is an actual reference to the instance of the page. It can be thought of as an object that represents the entire JSP page.

The page object is really a direct synonym for the **this** object.

The exception Object

The exception object is a wrapper containing the exception thrown from the previous page. It is typically used to generate an appropriate response to the error condition.

Servlet

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

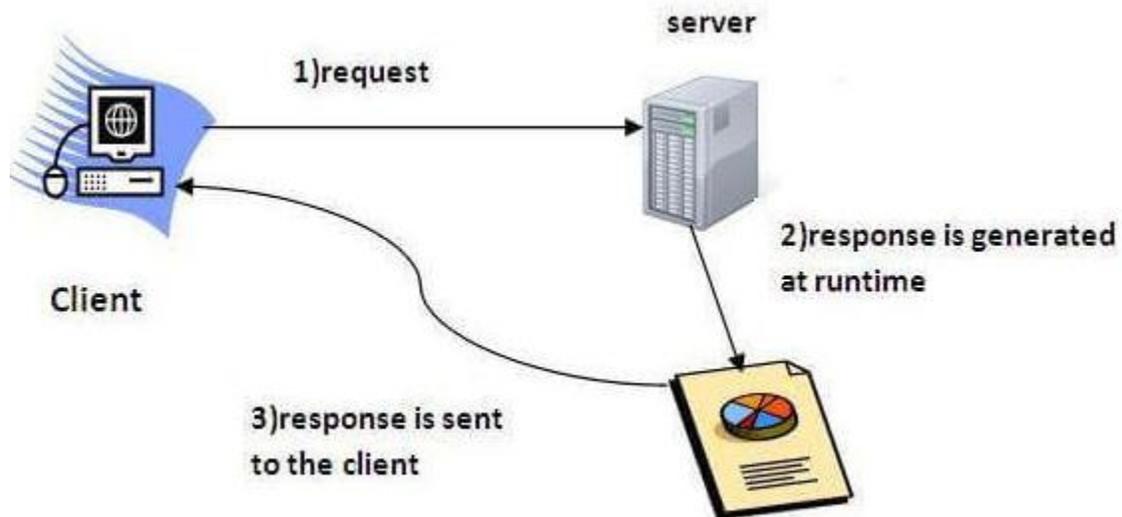
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.



What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. [C](#), [C++](#), [perl](#).

Advantages of Servlet

There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** [JVM](#) manages Servlets, so we don't need to worry about the memory leak, [garbage collection](#), etc.
4. **Secure:** because it uses java language.

Website

Website is a collection of related web pages that may contain text, images, audio and video. The first page of a website is called home page. Each website has specific internet address (URL) that you need to enter in your browser to access a website.

Website is hosted on one or more servers and can be accessed by visiting its homepage using a computer network. A website is managed by its owner that can be an individual, company or an organization.

A website can be of two types:

- Static Website
- Dynamic Website

Static website

Static website is the basic type of website that is easy to create. You don't need the knowledge of web programming and database design to create a static website. Its web pages are coded in HTML.

The codes are fixed for each page so the information contained in the page does not change and it looks like a printed page.

Dynamic website

Dynamic website is a collection of dynamic web pages whose content changes dynamically. It accesses content from a database or Content Management System (CMS). Therefore, when you alter or update the content of the database, the content of the website is also altered or updated.

Dynamic website uses client-side scripting or server-side scripting, or both to generate dynamic content.

Client side scripting generates content at the client computer on the basis of user input. The web browser downloads the web page from the server and processes the code within the page to render information to the user.

In server side scripting, the software runs on the server and processing is completed in the server then plain pages are sent to the user.

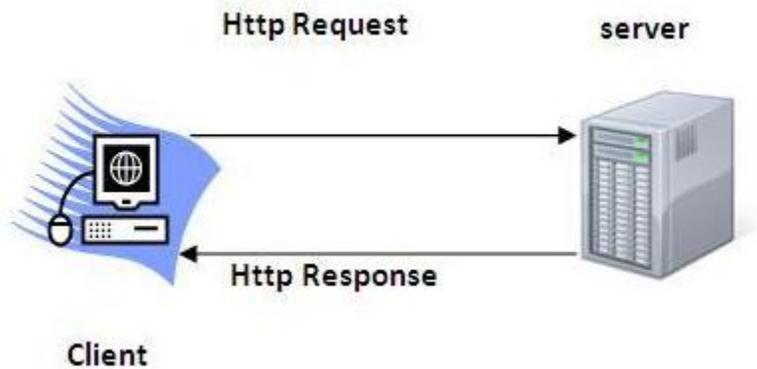
Static vs Dynamic website

| Static Website | Dynamic Website |
|---|---|
| Prebuilt content is same every time the page is loaded. | Content is generated quickly and changes re |
| It uses the HTML code for developing a website. | It uses the server side languages such as PHP and ASP.NET etc. for developing a website |
| It sends exactly the same response for every request. | It may generate different HTML for each of t |
| The content is only changed when someone publishes and updates the file (sends it to the web server). | The page contains "server-side" code which to generate the unique content when the pa |
| Flexibility is the main advantage of static website. | Content Management System (CMS) is the r dynamic website. |

HTTP (Hyper Text Transfer Protocol)

The Hypertext Transfer Protocol (HTTP) is application-level protocol for collaborative, distributed, hypermedia information systems. It is the data communication protocol used to establish communication between client and server.

HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.



The Basic Characteristics of HTTP (Hyper Text Transfer Protocol):

- It is the protocol that allows web servers and browsers to exchange data over the web.
- It is a request response protocol.
- It uses the reliable TCP connections by default on TCP port 80.
- It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.

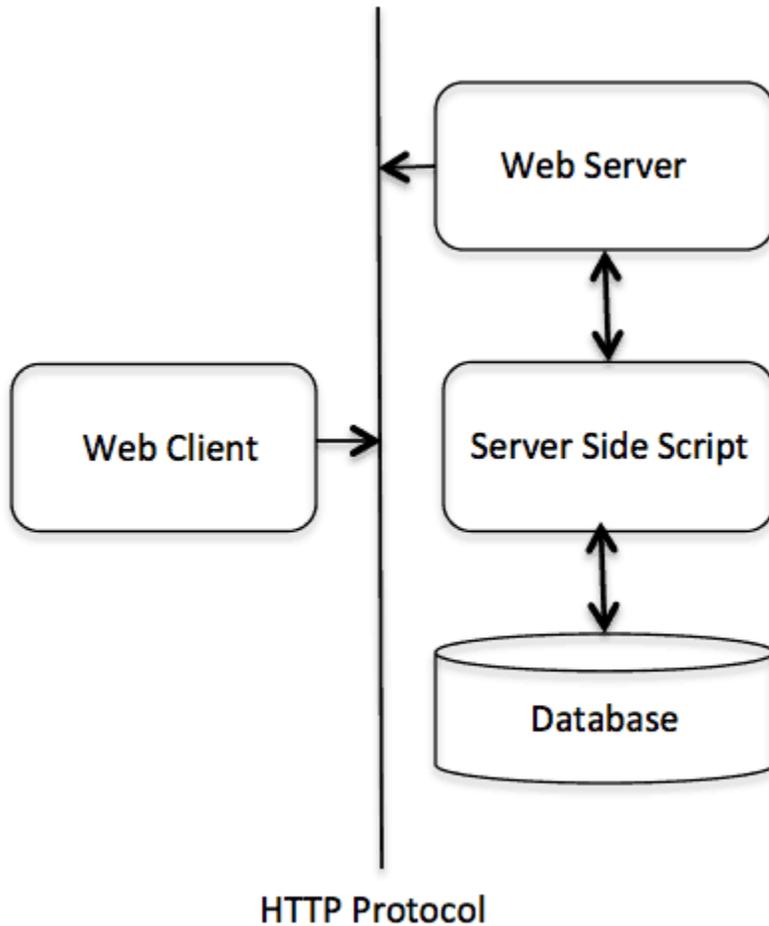
The Basic Features of HTTP (Hyper Text Transfer Protocol):

There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:

- **HTTP is media independent:** It specifies that any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.
- **HTTP is connectionless:** It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sent the client disconnects from server and waits for the response.
- **HTTP is stateless:** The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

The Basic Architecture of HTTP (Hyper Text Transfer Protocol):

The below diagram represents the basic architecture of web application and depicts where HTTP stands:



HTTP is request/response protocol which is based on client/server based architecture. In this protocol, web browser, search engines, etc. behave as HTTP clients and the Web server like Servlet behaves as a server

HTTP Requests

The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.

The HTTP client sends the request to the server in the form of request message which includes following information:

- The Request-line
- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)

- The Request method and Content
- The User-Agent header
- The Connection control header
- The Cache control header
- The HTTP request methods are:



| HTTP Request | Description |
|---------------------|---|
| GET | Asks to get the resource at the requested URL. |
| POST | Asks the server to accept the body info attached. It is like GET request with extra info request. |
| HEAD | Asks for only the header part of whatever a GET would return. Just like GET but with no body. |
| TRACE | Asks for the loopback of the request message, for testing or troubleshooting. |
| PUT | Says to put the enclosed info (the body) at the requested URL. |
| DELETE | Says to delete the resource at the requested URL. |
| OPTIONS | Asks for a list of the HTTP methods to which the thing at the request URL can respond. |

Unit – V

VB Script

VBScript stands for **V**isual **B**asic Scripting that forms a subset of Visual Basic for Applications (VBA). VBA is a product of Microsoft which is included NOT only in other Microsoft products such as MS Project and MS Office but also in Third Party tools such as AUTO CAD.

Features of VBScript

- VBScript is a lightweight scripting language, which has a lightning fast interpreter.
- VBScript, for the most part, is case insensitive. It has a very simple syntax, easy to learn and to implement.
- Unlike C++ or Java, VBScript is an object-based scripting language and NOT an Object-Oriented Programming language.
- It uses Component Object Model (**COM**) in order to access the elements of the environment in which it is executing.
- Successful execution of VBScript can happen only if it is executed in Host Environment such as Internet Explorer (**IE**), Internet Information Services (**IIS**) and Windows Scripting Host (**WSH**)

Disadvantages

- VBscript is used only by IE Browsers. Other browsers such as Chrome, Firefox DONOT Support VBScript. Hence, JavaScript is preferred over VBScript.
- VBScript has a Limited command line support.
- Since there is no development environment available by default, debugging is difficult.

Your First VBScript

Let us write a VBScript to print out "Hello World".

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

In the above example, we called a function *document.write*, which writes a string into the HTML document. This function can be used to write text, HTML or both. So, above code will display following result –

Hello World!

Whitespace and Line Breaks

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

Formatting

VBScript is based on Microsoft's Visual Basic. Unlike JavaScript, no statement terminators such as semicolon is used to terminate a particular statement.

Single Line Syntax

Colons are used when two or more lines of VBScript ought to be written in a single line. Hence, in VBScript, Colons act as a line separator.

```
<script language = "vbscript" type = "text/vbscript">  
  var1 = 10 : var2 = 20  
</script>
```

Multiple Line Syntax

When a statement in VBScript is lengthy and if user wishes to break it into multiple lines, then the user has to use underscore "_". This improves the readability of the code. The following example illustrates how to work with multiple lines.

```
<script language = "vbscript" type = "text/vbscript">  
  var1 = 10  
  var2 = 20  
  Sum = var1 + var2  
  document.write("The Sum of two numbers"&_"var1 and var2 is " & Sum)  
</script>
```

Reserved Words

The following list shows the reserved words in VBScript. These reserved words SHOULD NOT be used as a constant or variable or any other identifier names.

| | | |
|------|---------|----------|
| Loop | LSet | Me |
| Mod | New | Next |
| Not | Nothing | Null |
| On | Option | Optional |

| | | |
|---------|------------|------------|
| Or | ParamArray | Preserve |
| Private | Public | RaiseEvent |
| ReDim | Rem | Resume |
| RSet | Select | Set |
| Shared | Single | Static |
| Stop | Sub | Then |
| To | True | Type |
| And | As | Boolean |
| ByRef | Byte | ByVal |
| Call | Case | Class |
| Const | Currency | Debug |
| Dim | Do | Double |
| Each | Else | ElseIf |
| Empty | End | EndIf |
| Enum | Eqv | Event |

| | | |
|-----------------|-----------|---------------|
| Exit | False | For |
| Function | Get | GoTo |
| If | Imp | Implements |
| In | Integer | Is |
| Let | Like | Long |
| TypeOf | Until | Variant |
| Wend | While | With |
| Xor | Eval | Execute |
| Msgbox | Erase | ExecuteGlobal |
| Option Explicit | Randomize | SendKeys |

Case Sensitivity

VBScript is a **case-insensitive language**. This means that language keywords, variables, function names and any other identifiers need NOT be typed with a consistent capitalization of letters. So identifiers `int_counter`, `INT_Counter` and `INT_COUNTER` have the same meaning within VBScript.

Comments in VBScript

Comments are used to document the program logic and the user information with which other programmers can seamlessly work on the same code in future. It can include information such as developed by, modified by and it can also include incorporated logic. Comments are ignored by the interpreter while execution. Comments in VBScript are denoted by two methods.

1. Any statement that starts with a Single Quote (‘) is treated as comment.

Following is the example –

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

```
<!--  
' This Script is invoked after successful login  
' Written by : TutorialsPoint  
' Return Value : True / False  
// ->  
</script>
```

2. Any statement that starts with the keyword “REM”.

Following is the example –

```
<script language = "vbscript" type = "text/vbscript">  
<!--  
  REM This Script is written to Validate the Entered Input  
  REM Modified by : Tutorials point/user2  
  // ->  
</script>
```

VBScript Placement in HTML File

There is a flexibility given to include VBScript code anywhere in an HTML document. But the most preferred way to include VBScript in your HTML file is as follows –

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

In the following section, we will see how we can put VBScript in different ways –

VBScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows –

```
<html>  
<head>  
  <script type = "text/Vbscript">  
    <!--  
      Function sayHello()  
        MsgBox("Hello World")  
      End Function  
    //-->  
  </script>  
</head>  
  
<body>
```

```
<input type = "button" onclick = "sayHello()" value = "Say Hello" />
</body>
</html>
```

It will produce the following result – A button with the name SayHello. Upon clicking on the Button, the message box is displayed to the user with the message "Hello World".

VBScript in <body>...</body> section

If you need a script to run as the page loads so that the script generates content in the page, the script goes in the <body> portion of the document. In this case, you would not have any function defined using VBScript –

```
<html>
<head> </head>
<body>
  <script type = "text/vbscript">
    <!--
      document.write("Hello World")
    //-->
  </script>
  <p>This is web page body </p>
</body>
</html>
```

This will produce the following result –

Hello World

This is web page body

VBScript in <body> and <head> sections

You can put your VBScript code in <head> and <body> section altogether as follows –

```
<html>
<head>
  <script type = "text/vbscript">
    <!--
      Function sayHello()
        msgbox("Hello World")
      End Function
    //-->
  </script>
</head>

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

```
//-->
</script>
<input type = "button" onclick = "sayHello()" value = "Say Hello" />
</body>
</html>
```

It will produce the following result – Hello World message with a 'Say Hello' button. Upon Clicking on the button a message box with a message "Hello World" is displayed to the user.

Hello World

VBScript in External File

As you begin to work more extensively with VBScript, you will likely find that there are cases, where you are reusing identical VBScript code on multiple pages of a site. You are not restricted to be maintaining identical code in multiple HTML files.

The *script* tag provides a mechanism to allow you to store VBScript in an external file and then include it into your HTML files. Here is an example to show how you can include an external VBScript file in your HTML code using *script* tag and its *src* attribute –

```
<html>
<head>
  <script type = "text/vbscript" src = "filename.vbs" ></script>
</head>
<body>
  .....
</body>
</html>
```

To use VBScript from an external file source, you need to write your all VBScript source code in a simple text file with extension ".vbs" and then include that file as shown above. For example, you can keep the following content in filename.vbs file and then you can use *sayHello* function in your HTML file after including filename.vbs file.

```
Function sayHello()
  MsgBox "Hello World"
End Function
```

VBScript Placement in QTP

VBScript is placed in QTP (Quick Test Professional) tool but it is NOT enclosed within HTML Tags. The Script File is saved with the extension .vbs and it is executed by Quick Test Professional execution engine.

VBScript Variables

A variable is a named memory location used to hold a value that can be changed during the script execution. VBScript has only **ONE** fundamental data type, **Variant**.

Rules for Declaring Variables –

- Variable Name must begin with an alphabet.
- Variable names cannot exceed 255 characters.
- Variables Should NOT contain a period (.)
- Variable Names should be unique in the declared context.

Declaring Variables

Variables are declared using “dim” keyword. Since there is only ONE fundamental data type, all the declared variables are variant by default. Hence, a user **NEED NOT** mention the type of data during declaration.

Example 1 – In this Example, IntValue can be used as a String, Integer or even arrays.

Dim Var

Example 2 – Two or more declarations are separated by comma(,)

Dim Variable1,Variable2

Assigning Values to the Variables

Values are assigned similar to an algebraic expression. The variable name on the left hand side followed by an equal to (=) symbol and then its value on the right hand side.

Rules

- The numeric values should be declared without double quotes.
- The String values should be enclosed within double quotes("")
- Date and Time variables should be enclosed within hash symbol(#)

Examples

' Below Example, The value 25 is assigned to the variable.

Value1 = 25

' A String Value 'VBScript' is assigned to the variable StrValue.

StrValue = "VBScript"

' The date 01/01/2020 is assigned to the variable DToday.

Date1 = #01/01/2020#

' A Specific Time Stamp is assigned to a variable in the below example.

Time1 = #12:30:44 PM#

Scope of the Variables

Variables can be declared using the following statements that determines the scope of the variable. The scope of the variable plays a crucial role when used within a procedure or classes.

- Dim

- Public
- Private

Dim

Variables declared using “Dim” keyword at a Procedure level are available only within the same procedure. Variables declared using “Dim” Keyword at script level are available to all the procedures within the same script.

Example – In the below example, the value of Var1 and Var2 are declared at script level while Var3 is declared at procedure level.

Note – The scope of this chapter is to understand Variables. Functions would be dealt in detail in the upcoming chapters.

```
<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Dim Var1
      Dim Var2

      Call add()
      Function add()
        Var1 = 10
        Var2 = 15
        Dim Var3
        Var3 = Var1 + Var2
        MsgBox Var3 'Displays 25, the sum of two values.
      End Function

      MsgBox Var1 ' Displays 10 as Var1 is declared at Script level
      MsgBox Var2 ' Displays 15 as Var2 is declared at Script level
      MsgBox Var3 ' Var3 has No Scope outside the procedure. Prints Empty
    </script>
  </body>
</html>
```

Public

Variables declared using "Public" Keyword are available to all the procedures across all the associated scripts. When declaring a variable of type "public", Dim keyword is replaced by "Public".

Example – In the following example, Var1 and Var2 are available at script level while Var3 is available across the associated scripts and procedures as it is declared as Public.

```
<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
```

```

Dim Var1
Dim Var2
Public Var3

Call add()

Function add()
  Var1 = 10
  Var2 = 15
  Var3 = Var1+Var2
  MsgBox Var3 'Displays 25, the sum of two values.
End Function

Msgbox Var1 ' Displays 10 as Var1 is declared at Script level
Msgbox Var2 ' Displays 15 as Var2 is declared at Script level
Msgbox Var3 ' Displays 25 as Var3 is declared as Public

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

```

Private

Variables that are declared as "Private" have scope only within that script in which they are declared. When declaring a variable of type "Private", Dim keyword is replaced by "Private".

Example – In the following example, Var1 and Var2 are available at Script Level. Var3 is declared as Private and it is available only for this particular script. Use of "Private" Variables is more pronounced within the Class.

```

<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Dim Var1
      Dim Var2
      Private Var3

      Call add()
      Function add()
        Var1 = 10
        Var2 = 15
        Var3 = Var1+Var2
        MsgBox Var3 'Displays the sum of two values.
      End Function

      MsgBox Var1 ' Displays 10 as Var1 is declared at Script level
      MsgBox Var2 ' Displays 15 as Var2 is declared at Script level
    </script>
  </body>
</html>

```

```
Msgbox Var3 ' Displays 25 but Var3 is available only for this script.
</script>
</body>
</html>
```

Constant is a named memory location used to hold a value that CANNOT be changed during the script execution. If a user tries to change a Constant Value, the Script execution ends up with an error. Constants are declared the same way the variables are declared.

Declaring Constants

Syntax

```
[Public | Private] Const Constant_Name = Value
```

The Constant can be of type Public or Private. The Use of Public or Private is Optional. The Public constants are available for all the scripts and procedures while the Private Constants are available within the procedure or Class. One can assign any value such as number, String or Date to the declared Constant.

Example 1

In this example, the value of pi is 3.14 and it displays the area of the circle in a message box.

```
<!DOCTYPE html>
<html>
<body>
<script language = "vbscript" type = "text/vbscript">
  Dim intRadius
  intRadius = 20
  const pi = 3.14
  Area = pi*intRadius*intRadius
  MsgBox Area

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

Example 2

The below example illustrates how to assign a String and Date Value to a Constant.

```
<!DOCTYPE html>
<html>
<body>
<script language = "vbscript" type = "text/vbscript">
  Const myString = "VBScript"
  Const myDate = #01/01/2050#
  MsgBox myString
```

```
Msgbox myDate

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

Example 3

In the below example, the user tries to change the Constant Value; hence, it will end up with an **Execution Error**.

```
<!DOCTYPE html>
<html>
<body>
<script language = "vbscript" type = "text/vbscript">
  Dim intRadius
  intRadius = 20
  const pi = 3.14
  pi = pi*pi    'pi VALUE CANNOT BE CHANGED.THROWS ERROR'
  Area = pi*intRadius*intRadius
  MsgBox Area

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

What is an operator?

Let's take an expression $4 + 5$ is equal to 9. Here, 4 and 5 are called **operands** and + is called the **operator**. VBScript language supports following types of operators –

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Concatenation Operators

The Arithmetic Operators

VBScript supports the following arithmetic operators –

Assume variable A holds 5 and variable B holds 10, then –

Show Examples

| Operator | Description | Example |
|----------|-------------|---------|
|----------|-------------|---------|

| | | |
|---|---|------------------------|
| + | Adds two operands | A + B will give 15 |
| - | Subtracts second operand from the first | A - B will give -5 |
| * | Multiply both operands | A * B will give 50 |
| / | Divide numerator by denominator | B / A will give 2 |
| % | Modulus Operator and remainder of after an integer division | B MOD A will give 0 |
| ^ | Exponentiation Operator | B ^ A will give 100000 |

To understand these operators in a better way, you can [Try it yourself](#).

The Comparison Operators

There are following comparison operators supported by VBScript language –

Assume variable A holds 10 and variable B holds 20, then –

[Show Examples](#)

| Operator | Description | Example |
|----------|---|--------------------|
| = | Checks if the value of two operands are equal or not, if yes then condition becomes true. | (A == B) is False. |
| <> | Checks if the value of two operands are equal or not, if values are not equal then condition becomes true. | (A <> B) is True. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is False. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is True. |

| | | |
|----|---|--------------------|
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is False. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is True. |

To understand these operators in a better way, you can [Try it yourself](#).

The Logical Operators

There are following logical operators supported by VBScript language –

Assume variable A holds 10 and variable B holds 0, then –

[Show Examples](#)

| Operator | Description | Example |
|----------|--|-----------------------------|
| AND | Called Logical AND operator. If both the conditions are True, then Expression becomes True. | a<>0 AND b<>0 is False. |
| OR | Called Logical OR Operator. If any of the two conditions is True, then condition becomes True. | a<>0 OR b<>0 is true. |
| NOT | Called Logical NOT Operator. It reverses the logical state of its operand. If a condition is True, then the Logical NOT operator will make it False. | NOT(a<>0 OR b<>0) is false. |
| XOR | Called Logical Exclusion. It is the combination of NOT and OR Operator. If one, and only one, of the expressions evaluates to True, result is True. | (a<>0 XOR b<>0) is true. |

To understand these operators in a better way, you can [Try it yourself](#).

The Concatenation Operators

There are following Concatenation operators supported by VBScript language –

Assume variable A holds 5 and variable B holds 10 then –

Show Examples

| Operator | Description | Example |
|-----------------|--|---------------------|
| + | Adds two Values as Variable Values are Numeric | A + B will give 15 |
| & | Concatenates two Values | A & B will give 510 |

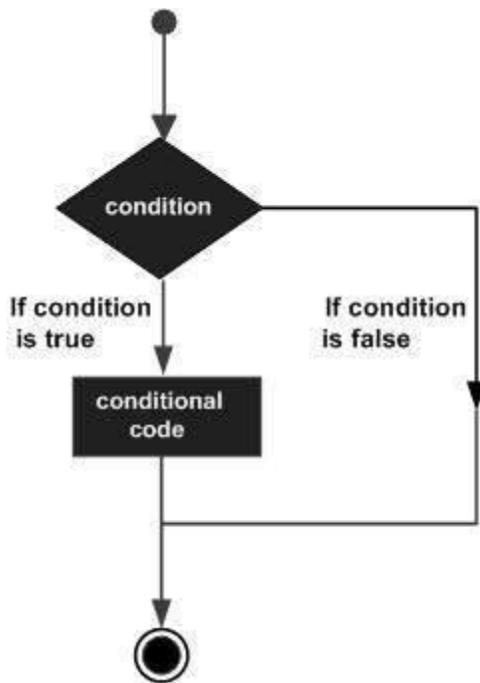
Assume variable A = "Microsoft" and variable B="VBScript", then –

| Operator | Description | Example |
|-----------------|-------------------------|-----------------------------------|
| + | Concatenates two Values | A + B will give MicrosoftVBScript |
| & | Concatenates two Values | A & B will give MicrosoftVBScript |

Decision making

Decision making allows programmers to control the execution flow of a script or one of its sections. The execution is governed by one or more conditional statements.

Following is the general form of a typical decision making structure found in most of the programming languages –



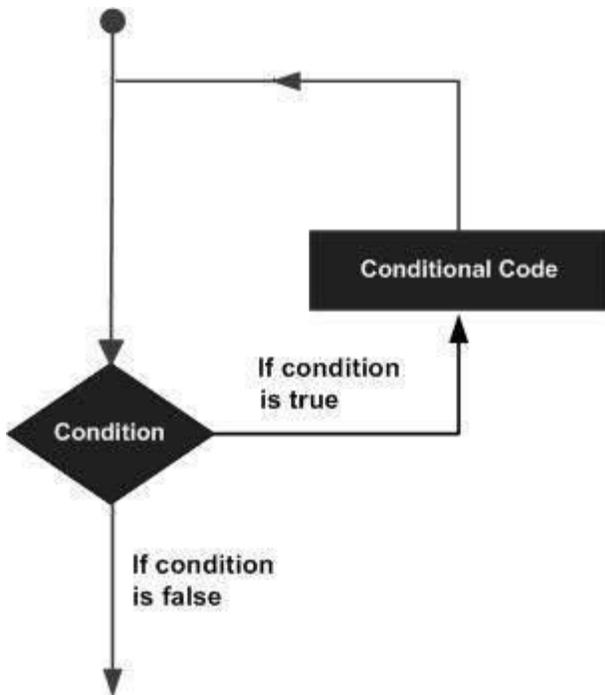
VBScript provides the following types of decision making statements.

| Statement | Description |
|-----------------------------|---|
| if statement | An if statement consists of a Boolean expression followed by one or more statements. |
| if..else statement | An if else statement consists of a Boolean expression followed by one or more statements. If the condition is True, the statements under the If statements are executed. If the condition is false, then the Else part of the script is Executed |
| if...elseif..else statement | An if statement followed by one or more ElseIf Statements, that consists of Boolean expressions and then followed by an optional else statement, which executes when all the condition becomes false. |
| nested if statements | An if or elseif statement inside another if or elseif statement(s). |
| switch statement | A switch statement allows a variable to be tested for equality against a list of values. |

Loop

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in VBScript.



VBScript provides the following types of loops to handle looping requirements. Click the following links to check their detail.

| Loop Type | Description |
|---|---|
| <u>for loop</u> | Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| <u>for ..each loop</u> | It is executed if there is at least one element in group and reiterated for each element in a group. |
| <u>while..wend loop</u> | It tests the condition before executing the loop body. |

| | |
|--|---|
| <u>do..while loops</u> | The do..While statements will be executed as long as condition is True.(i.e.,) The Loop should be repeated till the condition is False. |
| <u>do..until loops</u> | The do..Until statements will be executed as long as condition is False.(i.e.,) The Loop should be repeated till the condition is True. |

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all the remaining statements in the loop are NOT executed.

VBScript supports the following control statements. Click the following links to check their detail.

| Control Statement | Description |
|---|--|
| <u>Exit For statement</u> | Terminates the For loop statement and transfers execution to the statement immediately following the loop |
| <u>Exit Do statement</u> | Terminates the Do While statement and transfers execution to the statement immediately following the loop |

Function

What is a Function?

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing same code over and over again. This will enable programmers to divide a big program into a number of small and manageable functions. Apart from inbuilt Functions, VBScript allows us to write user-defined functions as well. This section will explain you how to write your own functions in VBScript.

Function Definition

Before we use a function, we need to define that particular function. The most common way to define a function in VBScript is by using the Function keyword, followed by a unique function name and it may or may not carry a list of parameters and a statement with an **End Function** keyword, which indicates the end of the function.

The basic syntax is shown below –

```
<!DOCTYPE html>
```

```

<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Function Functionname(parameter-list)
        statement 1
        statement 2
        statement 3
        .....
        statement n
      End Function

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

```

Example

```

<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Function sayHello()
        msgbox("Hello there")
      End Function

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

```

Calling a Function

To invoke a function somewhere later in the script, you would simple need to write the name of that function with the **Call** keyword.

```

<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Function sayHello()
        msgbox("Hello there")
      End Function

      Call sayHello()

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

```

Function Parameters

Till now, we have seen function without a parameter, but there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. The Functions are called using the **Call** Keyword.

```
<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Function sayHello(name, age)
        msgbox( name & " is " & age & " years old.")
      End Function

      Call sayHello("Tutorials point", 7)

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

Returning a Value from a Function

A VBScript function can have an optional return statement. This is required if you want to return a value from a function. For example, you can pass two numbers in a function and then you can expect from the function to return their multiplication in your calling program.

NOTE – A function can return multiple values separated by comma as an array assigned to the function name itself.

Example

This function takes two parameters and concatenates them and returns result in the calling program. In VBScript, the values are returned from a function using function name. In case if you want to return two or more values, then the function name is returned with an array of values. In the calling program, the result is stored in the result variable.

```
<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Function concatenate(first, last)
        Dim full
        full = first & last
        concatenate = full 'Returning the result to the function
name itself
      End Function

    </script>
```

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

Now, we can call this function as follows –

```
<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Function concatenate(first, last)
        Dim full
        full = first & last
        concatenate = full 'Returning the result to the function
name itself
      End Function
      ' Here is the usage of returning value from function.
      dim result
      result = concatenate("Zara", "Ali")
      msgbox(result)
    </script>
  </body>
</html>
```

Sub Procedures

Sub-Procedures are similar to functions but there are few differences.

- Sub-procedures DONOT Return a value while functions may or may not return a value.
- Sub-procedures Can be called without call keyword.
- Sub-procedures are always enclosed within **Sub** and **End Sub** statements.

Example

```
<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Sub sayHello()
        msgbox("Hello there")
      End Sub
    </script>
  </body>
</html>
```

Calling Procedures

To invoke a Procedure somewhere later in the script, you would simply need to write the name of that procedure with or without the **Call** keyword.

```
<!DOCTYPE html>
<html>
  <body>
    <script language = "vbscript" type = "text/vbscript">
      Sub sayHello()
        msgbox("Hello there")
      End Sub
      sayHello()
    </script>
  </body>
</html>
```

Advanced Concepts for Functions

There is a lot to learn about VBScript functions. We can pass the parameter byvalue or byreference. Please click on each one of them to know more.

- [ByVal- Pass the parameters by value](#)
- [ByRef- Pass the parameters by the reference](#)