UNIT I WEBSITE BASICS, HTML 5, CSS 3, WEB 2.0 7

Web Essentials: Clients, Servers and Communication – The Internet – World wide web – HTTP Request Message – HTTP Response Message – Web Clients – Web Servers – HTML5 – Tables – Lists – Image – HTML5 control elements – Drag and Drop – Audio – Video controls - CSS3 – Inline, embedded and external style sheets – Rule cascading – Inheritance – Backgrounds – Border Images – Colors – Shadows – Text – Transformations – Transitions – Animations. Bootstrap Framework

1.1: Web Essentials

Server:

The software that distributes the information and the machine where the information and software reside is called the server.

- ✓ provides requested service to client
- ✓ e.g., Web server sends requested Web page

Client:

The software that resides on the remote machine, communicates with the server, fetches the information, processes it, and then displays it on the remote machine is called the client.

- ✓ initiates contact with server (speaks first)
- ✓ typically requests service from server
- ✓ Web client is implemented in browser

Web server: Software that delivers Web pages and other documents to browsers using the HTTP protocol.

Web Page: A web page is a document or resource of information that is suitable for the World Wide Web and can be accessed through a web browser.

Website: A collection of pages on the World Wide Web those are accessible from the same URL and typically residing on the same server.

URL: Uniform Resource Locator, the unique address which identifies aresource on the Internet for routing purposes.

Below is additional information about each of the sections of the httpURL for this page.

CLIENT-SERVER PARADIGM

- ✓ The Client-Server paradigm is the most prevalent model for distributed computing protocols. It is the basis of all distributed computing paradigms at a higher level of abstraction.
- ✓ It is service-oriented, and employs a request-response protocol.

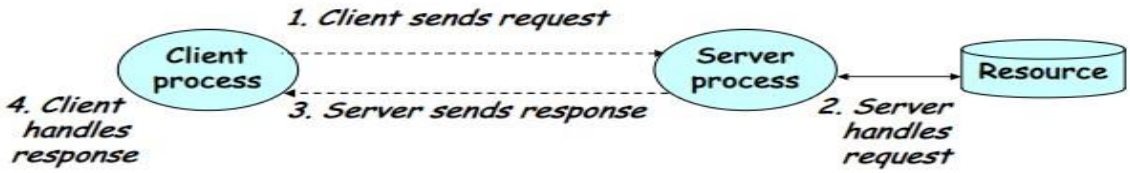
- A server process, running on a server host, provides access to a service. A client process, running on a client host, accesses the service via the server process.
- The interaction of the process proceeds according to a protocol.
- The primary idea of a client/server system is that you have a central repository of information—some kind of data, often in a database—that you want to distribute on demand to some set of people or machines.

Clients

- Examples of client programs
  - Web browsers, `ftp`, `telnet`, `ssh`
- How does a client find the server?
  - The IP address in the server socket address identifies the host
  - The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.
  - Examples of well known ports
    - Port 7: Echo server
    - Port 23: Telnet server
    - Port 25: Mail server
    - Port 80: Web server

Client/sever model

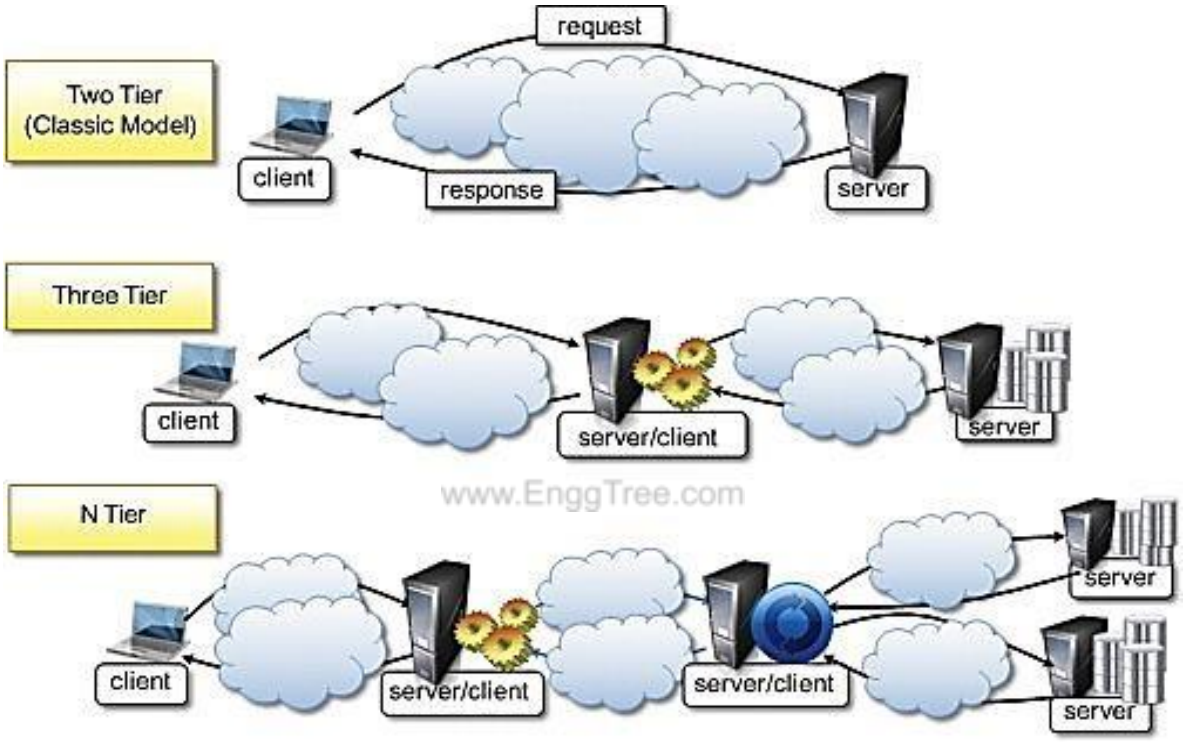
- Client asks (*request*) – server provides (*response*)
- Typically: single server - multiple clients
- The server does not need to know *anything* about the client
  - even that it exists
- The client should always know *something* about the server
  - at least where it is located



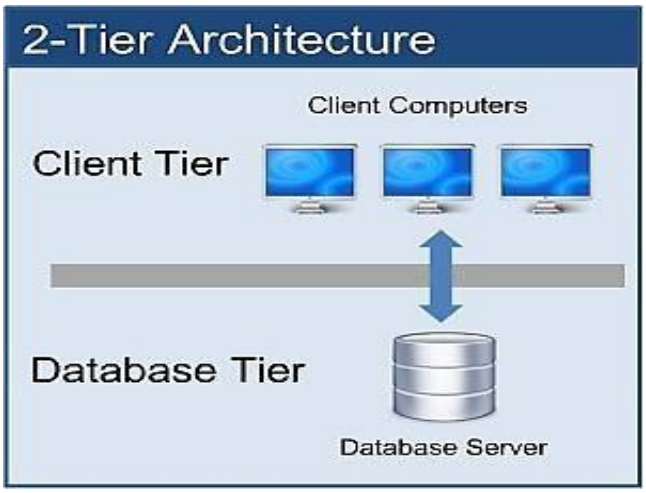
Note: clients and servers are processes running on hosts (can be the same or different hosts).

TYPES OF CLIENT-SERVER ARCHITECTURE

- Depending upon the number of layers or tiers used in the client-server model, it can be categorized into 3 types:
  - ✓ 2 Tier Architecture
  - ✓ 3Tier Architecture
  - ✓ N Tier Architecture



1) Two Tiers Architecture



- ❖ In this type of architecture, the workload is divided between the server (host of the system) and the client(which hosts the user interface).
- ❖ In reality these are located on separate computers but there is no absolute requirement of

this, providing that the tiers are logically separated can be hosted (e.g. development and testing) on the same computer.

Advantages:

- ✓ Ease in Developing Applications:
- ✓ User Satisfaction:
- ✓ Applicable for Homogeneous Environment:
- ✓ High Performance:

Limitations:

The two tier architecture proved to be a good solution when user population work is usually small (up to about 100 concurrent users) but it rapidly proved to have a number of limitations:

- ✓ **Performance:** Performance begins to deteriorate as the population grows.
- ✓ **Security:** Each user must have their own individual access to the database, and be granted whatever rights may be required in order to run the application.
- ✓ **Capability:** Independent of the type of client, much of the data processing has to be located in database making it totally dependent upon the capabilities and implementation provided by the database manufacturer.
- ✓ **Portability:** As the two-tier architecture is dependent upon the specific database implementation, porting an existing application to a different DBMS becomes a major issue.

1) Three-Tier Architecture

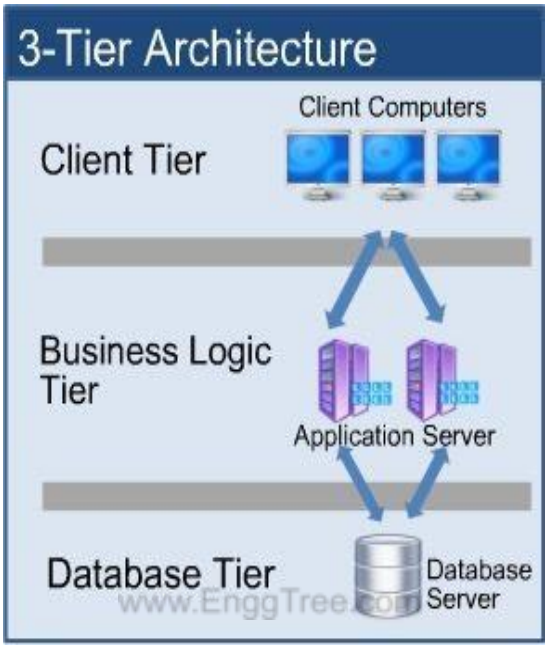
In the three-tier architecture the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on separate platform.

The three-tiers in a three-tier architecture are:

(i) Presentation Tier: Occupies the top level and displays information related to service available on a website. This tier communicates with other tiers by sending results to the browser and other tiers in the network.

(ii) Application Tier: Also called the middle tier, logic tier or business logic, this tier is pulled from the presentation tier. It controls application functionality by performing detailed processing.

(iii) Data Tier: Houses database server where information is stored and retrieved. Data in this tier is kept independent of application servers or business logic.



Advantages:

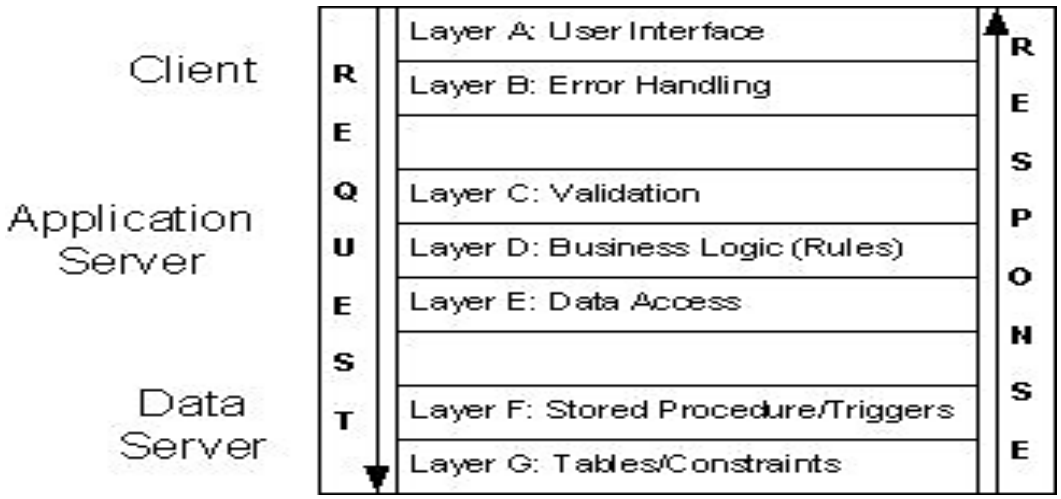
- ✓ **Improved Data Integrity**
- ✓ **Enhanced Security**
- ✓ **Hidden Database Structure**

Limitations:

- ❖ **Complexity of Communication:** Usually more efforts should be ensured when creating 3-tier applications as the communication points are increased (client to middle tier to server) and the performance increased by tools like Visual Basics, Power Builder etc.

2) N-Tier Architecture

N Tier Architecture often referred as Multitier Architecture. It is client-server architecture in which presentation, application processing, and data management functions are physically separated. it is an expanded form of three-tierarchitecture.



Advantages:

- ✓ It provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developer acquires the option of modifying or adding a specific layer, instead of reworking the application.

Limitations:

- ❖ Difficult to Implement: Due to componentization of tiers, the complex structure is difficult to implement or maintain.

1.2: The Internet

❖ **INTERNET:**

The Internet is a massive network of networks that connects millions of computers together globally. It forms a network in which any computer can communicate with any other computer with the help of TCP/IP protocol as long as they are both connected to the Internet.

❖ **IP ADDRESSING:**

Each host on a TCP/IP network is assigned a unique 32 bit logical address and it is called as Internet Protocol Address (IP Address). IP address is used to identify a particular host on the internet.

- ✓ Each IP address Consists of 4 bytes or 32 bits. This is represented in *quad notation* (or *dot* notation) as 4 x 8 bit numbers, each in the range 0 to 255, e.g. 131.123.2.220.
- ✓ IP address is divided into two main parts: 1. Network Number
  2. Host Number

✓ **5 classes of IP address:**

IP address class	Format	Range	Purpose
Class A	N.H.H.H	1 to 126	Large organization
Class B	N.N.H.H	127 to 191	Medium organization
Class C	N.N.N.H	192 to 223	Small organization
Class D	-	224 to 239	Multicast group
Class E	-	240 to 254	Experimental purpose

Loopback Address: (127.0.0.0 to 127.255.255.255)

Addresses within this range (127.0.0.0 to 127.255.255.255) are called loopback addresses used for loop back functionality. IP datagrams sent by a host to a 127.X.X.X loopback address are not passed down to the data link layer for transmission.

❖ **INTERNET SERVICE PROVIDER(ISP):**

An **Internet Service Provider** (**ISP**) is an organization that provides its customers the services for accessing, using, or participatingin the Internet.

Examples: BSNL, AirTel, Aircel etc.,

❖ **FIREWALLS:**

The hardware and software that sits between the internet and the local network, checking all the data that comes in and out to make sure that it is legitimate is called a **Firewall.**

The most basic firewall is a packet filter that inspects each packet coming in or out of a network and uses a set of rules to determine whether that traffic is allowed.

❖ **PROXY SERVERS:**

A proxy server is computer that functions as an intermediary between a web browser (such as Internet Explorer) and the Internet. Proxy servers help improve web performance by storing a copy of frequently used webpages. When a browser requests a webpage stored in the proxy server's collection (its cache), it is provided by the proxy server, which is faster than going to the web. Proxy servers also help improve security by filtering out some web content and malicious software.

❖ GATEWAYS:

Gateway is a machine used to route packets form one network to another network. In internet gateways provide all interconnections between physical networks.

Gateways have two responsibilities:

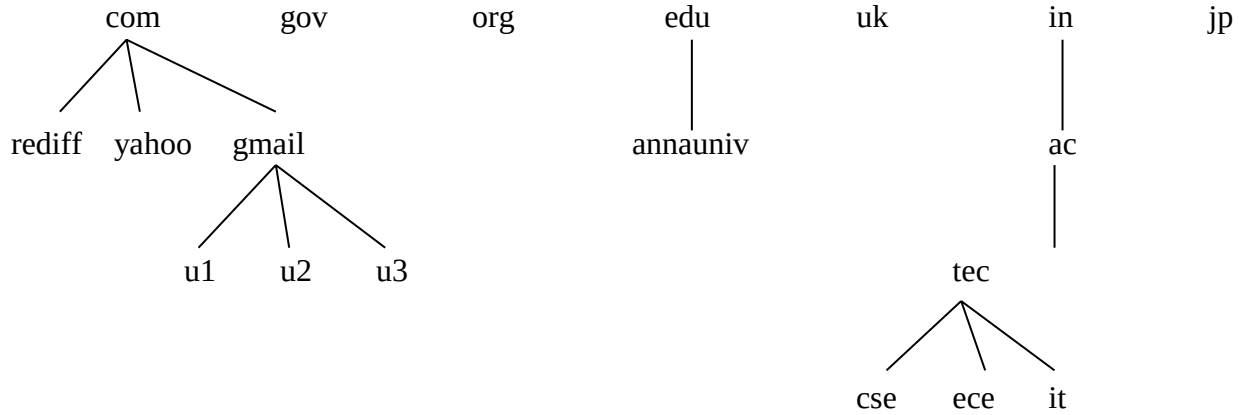
1. Route packets based on network id to a gateway connected to that network.
2. If they are connected to destination network, make sure that the packet gets delivered to correct machine on that network.

❖ DNS (DOMAIN NAME SERVICE):

DNS is application software that provides a mechanism for mapping back and forth between IP addresses and host names. DNS (Domain Name System) contains a large database of domain names and their correspondent Internet (IP Addresses) for example: www.widget.com corresponds to its unique number 207.168.6.12

- **Domain Names:** The *domain name* is the user-friendly equivalentof an IP address. Used because the numbers in an IP address are hard to remember and use. Also known as a host name. Example:**shu.ac.uk**
  - ✓ Internet host names consists f a sequence of labels separated by dots. The final label in a host name is a **top-level domain name.**
  - ✓ Two standard top-level domains are:
    1. Generic (.com, .edu, .org, .biz, .net, .mil)
    2. Country Code (.de, .il,. .uk, .in, .jp)
  - ✓ Top-level domain names are assigned by the **InternetCorporation for Assigned Names & Numbers (ICANN).**
  - ✓ Each top-level domain is divided into sub-domain (second level domains) assigned by registry authority of ICANN.

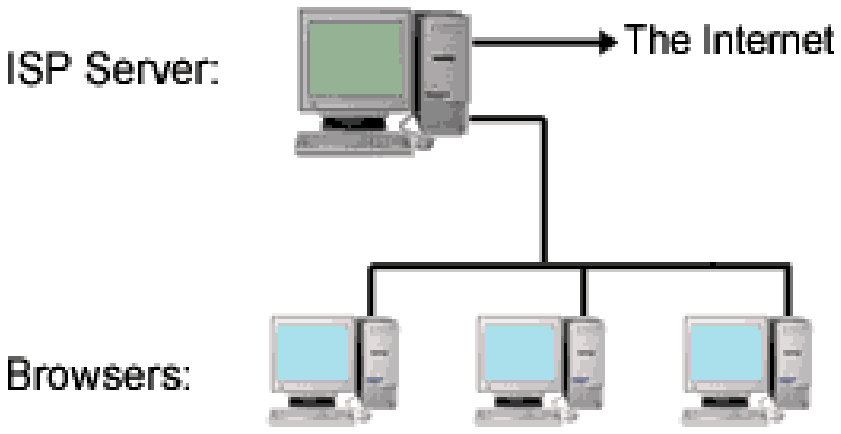
Domain Names



Uniquely traced with the help of domain name space **(http://www.cse.tec.ac.in)**

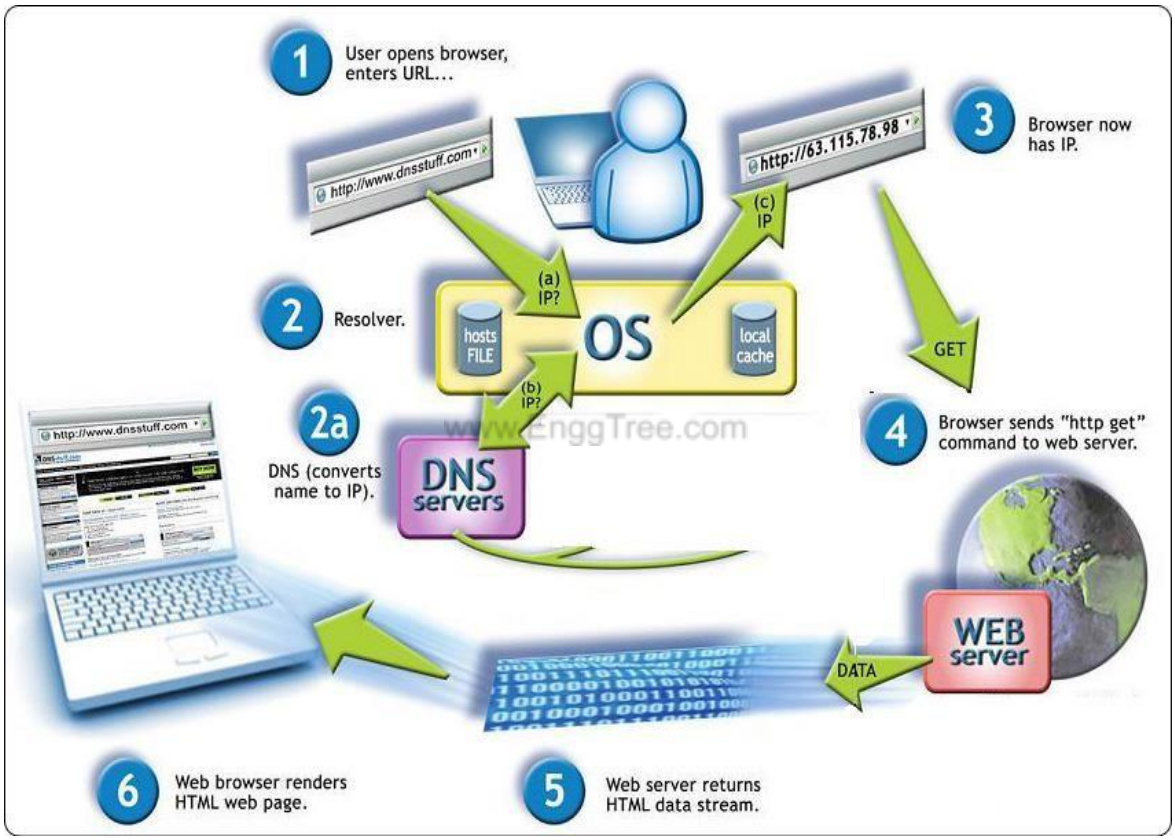
HOW INTERNET WORKS?

- ✓ The internet is a world-wide network of computers linked together by telephone wires, satellite links and other means.
- ✓ All computers on the internet can be divided into two categories: *servers* and *browsers*.
- ✓ **Servers** are where most of the information on the internet "lives". These are specialized computers which store information, share information with other servers, and make this information availableto the general public.
- ✓ **Browsers are** what people use to access the World Wide Web from any standard computer. Chances are, the browser you're using to view this page is either *Netscape Navigator/Communicator or Microsoft Internet Explorer*. These are by far the most popular browsers, but there are also a number of others in common use.
- ✓ When you connect your computer to the internet, you are connecting to a special type of server which is provided and operated by your Internet Service Provider (ISP). The job of this "ISP Server" is to provide the link between your browser and the rest of the internet. A single ISP server handles the internet connections ofmany individual browsers - there may be thousands of other people connected to the same server that you are connected to right now.
- ✓ The following picture shows a small "slice" of the internet with several home computers connected to a server:



To view a web page from your browser, the following sequence happens:

1. You either type an address (URL) into your "Address Bar" or click on a hyperlink.
2. Your browser sends a request to your ISP server asking for the page.
3. Your ISP server looks in a huge database of internet addresses and finds the exact host server which houses the website in question, then sends that host server a request for the page.
4. The host server sends the requested page to your ISP server.
5. Your ISP sends the page to your browser and you see it displayed on your screen.



❖ BASIC INTERNET PROTOCOLS

Definition: Protocol

A protocol can be defined as the set of rules governing the syntax, semantics and synchronization of communication between two end points.

Some common protocols used in Internet Communications:

1. TCP (Transmission Control Protocol)
2. UDP (User Datagram Protocol)
3. IP (Internet Protocol)
4. HTTP (Hyper Text Transfer Protocol)
5. FTP (File Transfer Protocol)
6. Email Protocols:
   - ✓ SMPT (Simple Mail Transfer Protocol)
   - ✓ POP3 (Post Office Protocol 3)
   - ✓ IMAP (Internet Message Access Protocol)
7. DHCP (Dynamic Host Configuration Protocol)

OSI Layer	Protocols used
Application	FTP, Telnet, SMTP, SNMP
Presentation	
Session	
Transport	TCP, UDP
Network	Routing Protocols – IP
Data Link	ARP, RARP
Physical	Ethernet

1. Transmission Control Protocol (TCP):

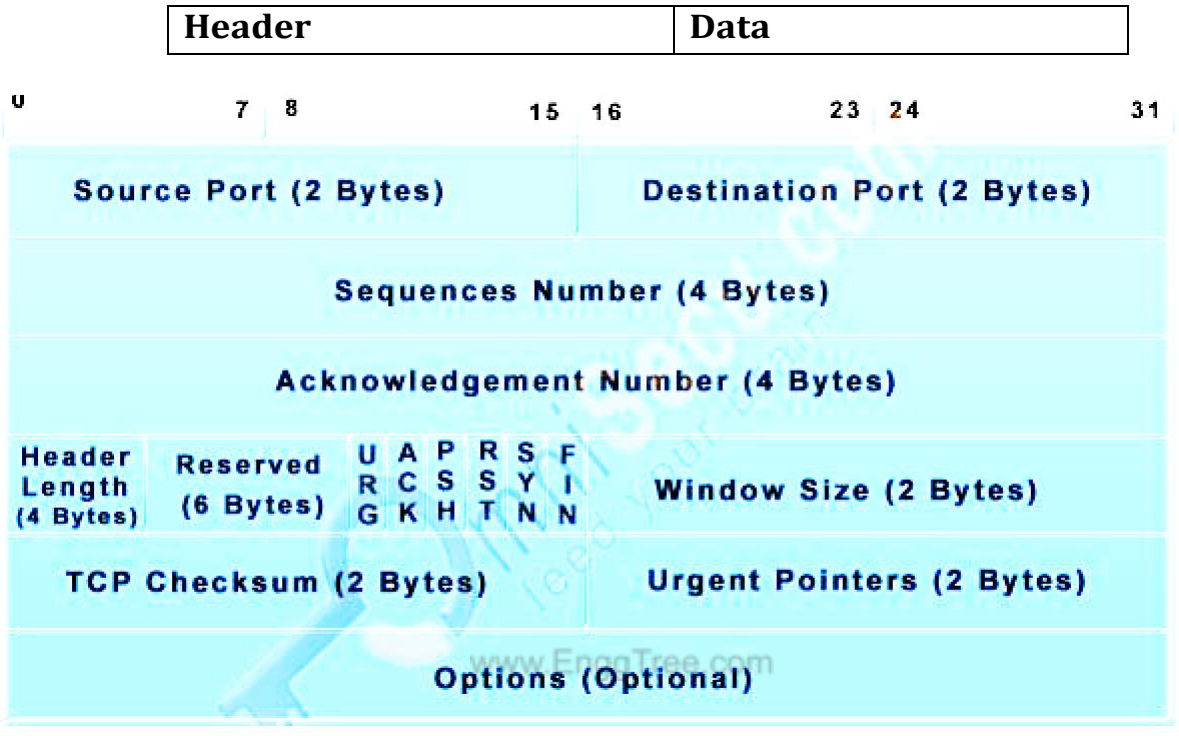
TCP is connection-oriented protocol that offers end-to-end packetdeliver. It is a reliable, transport layer protocol.

TCP Services:

- Stream delivery service – TCP is stream oriented because it allows the sending process to sen data as stream of bytes and the receiving process to receive data as stream of bytes.
- Sending and receiving buffers – It may not be possible for sender and the receiver process to produce and consume data at same speed. Therefore TCP requires buffer for storage at both the ends.
- Bytes and segments – TCP groups a number of bytes into apacket called **segments.**
- Full duplex service – Allows flow of data in both the directions.
- Efficient Flow Control
- Connection Oriented Service – Since it is connection-oriented protocol, before data transmission takes place both the sending and receiving

- ends must establish a connection between each other.
- Reliable Service – It uses the acknowledgement mechanism to ensure the safe and sound arrival of data.

TCP Segment Format:



The unit of data transfer between two devices using TCP is called as **Segment.** The segment consists of 20 to 60 byte header followed by data from the application program.

2. User Datagram Protocol (UDP):

- ✓ UDP is connectionless, unreliable transport layer protocol. It provides process-to-process communication instead of host-to- host communication.
- ✓ It performs very limited error checking.
- ✓ UDP is a very simple protocol with a minimum overhead. If a process wants to send a small message and does not care about reliability, it can use UDP.
- ✓ Sending a small message using UDP takes much less interaction between the sender and receiver than TCP.

UCP is convenient protocol for multimedia and multicasting applications.

UDP Header Format:

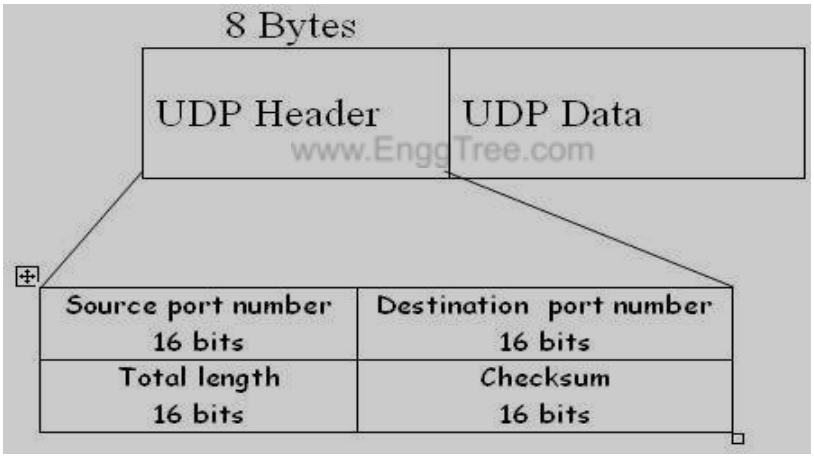
UDP packets, called user datagram, have a fixed size headerof 8 bytes.

Source Port Number: 16 bit port number used by the running process on the source host.

Destination Port Number: 16 bit port number used by the process running on the destination host.

Length: 16 bit field that defines the total length of the user datagram [header + data].

Checksum: This field is used to detect errors over the entire user datagram [header+data]. This ensures that the fields have not changed from the source to destination.



✓ **Applications:**

- UCP is suitable for process that requires simple request- response communication with little concern for flow controland error control.
- UDP is suitable for a process with internal flow and errorcontrol mechanism.
- UDP is suitable for multicasting.
- UDP is suitable for sending multimedia messages.

DIFFERENCE BETWEEN TCP AND UDP:

	TCP	UDP
Acronym for	Transmission Control Protocol	User Datagram Protocol or Universal Datagram Protocol
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Usage	TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	UDP is suitable for applications that need fast, efficient transmission, such as games. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients.
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no order as all packets are independent of each other. If ordering is required, it has to be managed by the application layer.
Speed of transfer	The speed for TCP is slower than UDP.	UDP is faster because there is no error-checking for packets.

Header Size	TCP header size is 20 bytes	UDP Header size is 8 bytes.
Common Header Fields	Source port, Destination port, Check Sum	Source port, Destination port, Check Sum
Weight	TCP is heavy-weight. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and Congestion control.	UDP is lightweight. There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.
Data Flow Control	TCP does Flow Control. TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.	UDP does not have an option for flow control
Error Checking	TCP does error checking	UDP does error checking, but no recovery options.
Acknowledgement	Acknowledgement segments	No Acknowledgment
Handshake	SYN, SYN-ACK, ACK	No handshake (connectionless protocol)

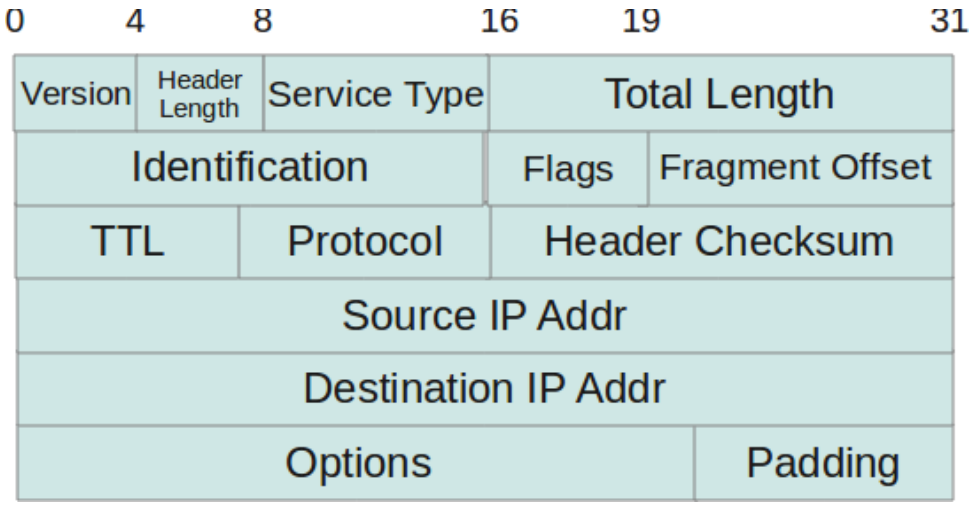
3. Internet Protocol(IP):

- ✓ The **Internet Protocol** (**IP**) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet.
- ✓ IP is the host-to-host network layer protocol for the internet.
- ✓ IP is connection less and unreliable protocol. It is connection less in the sense that no state related to IP datagrams is maintained either on source or destination side
- ✓ It is unreliable in the sense that it not guaranteed that an IP data gram will get delivered to the destination or not.
- ✓ If an IP datagram encounters some error at the destination or at some intermediate host (while traveling from source to destination) then the IP datagram is generally discarded and anICMP error message is sent back to the source.
- ✓ If reliability is important, it must be paired with a reliable protocol such as TCP.

IP Header Format:

Protocol Version(4 bits) : This is the first field in the protocol header. This field occupies 4 bits. This signifies the current IP protocol version being used.

Header Length(4 bits) *:* This field provides the length of the IP header. The length of the header is represented in 32 bit words.



Uses of IP Protocol:

Addressing

- While sending datagrams, an addressing mechanism is needed to send the datagrams accurately. In order to achieve this, IP uses a technique for host addressing. The addressing of devices (to which the datagrams are delivered) needs to be unique as this system needs to work across networks.

Routing

- When a datagram is sent from one network to another, which are distant and not directly connected, the delivery is indirect. IP supports this functionality by routing the datagram through intermediate devices (routers). It uses Internet Control Message Protocol (ICMP) and routing protocols like Routing Information Protocol (RIP) and Border Gateway Protocol (BGP) to achieve this. **DATA Encapsulation**
- IP provides security to networks by encapsulating the data into an IP datagram. This makes sure it is received and interpreted by the intended recipient.

Formatting/Packaging

- IP uses a specific formatting and packaging prior to transmission. IP accepts data from the transport layer protocols above it in the OSI layer--UDP and TCP--and passes them to the data layers. This format and package is only decipherable by the recipient.

Fragmentation

- Since the frame size of each physical and data link network using IP may be different, IP fragments datagrams into pieces, so that they can each be carried on the local network. This helps with network reliability.

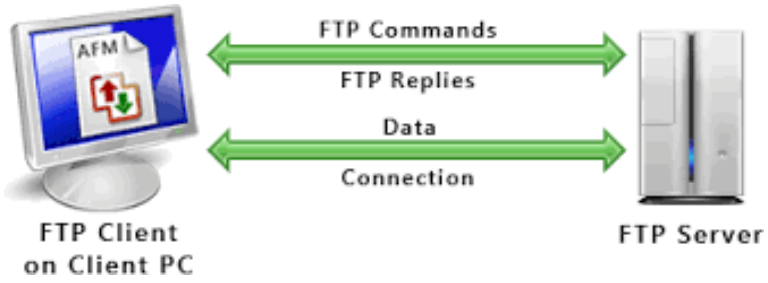
Reassembly

- IP reassembles the datagrams received into the full IP datagram (as they may be fragmented) for the receiving device and passes it on to the higher layers for interpretation.

4. File Transfer Protocol (FTP):

- ✓ File Transfer Protocol, or FTP, is a protocol used for transferring files from one computer to another - typically from your computer to a web server.
- ✓ FTP is the preferred method of exchanging files because it's faster than other protocols like HTTP or POP. If you need to exchange large files, you should consider FTP.

- ✓ FTP data is sent and received through computer port 21 and under the TCP protocol.
- ✓ The transfer is asynchronous, meaning not at the same time and therefore faster than other protocols.



Objectives of FTP were:

1. to promote sharing of files (computer programs and/or data),
2. to encourage indirect or implicit (via programs) use of remotecomputers,
3. to shield a user from variations in file storage systems amonghosts, and
4. to transfer data reliably and efficiently.

5. Hyper Text Transfer Protocol (HTTP):

- ✓ HTTP is a request/response protocol. It is a communication protocol used to transfer the information on local area network and WWW.
- ✓ HTTP (Hypertext Transfer Protocol) is the set of rules for transferring files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. As soon as a Web user opens their Web browser, the user is indirectly making use of HTTP. HTTP is an application protocol that runs on top of the TCP/IP suite of protocols (the foundation protocols for the Internet).
- ✓ HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.
- ✓ HTTP is called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it. Hence this protocol cannot maintain the previous conversations held between the browser and the server.

- ✓ Standard port for HTTP to listen on is 80.
- ✓ **GET** and **POST** are the two popular methods used by the HTTP protocol to make requests to the server on WWW.

6. Simple Mail Transfer Protocol (SMTP):

- ✓ SMTP (Simple Mail Transfer Protocol) is a TCP/IP protocol used in sending and receiving e-mail.
- ✓ However, since it is limited in its ability to queue messages at the receiving end, it is usually used with one of two other protocols, POP3 or IMAP that let the user save messages in a server mailbox and download them periodically from the server.
- ✓ In other words, users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving e-mail.
- ✓ SMTP is a application level protocol.
- ✓ SMTP is connection oriented protocol.
- ✓ It handles exchange of messages between e-mail servers over TCP/IP network.
- ✓ Default port of SMTP is 25.

7. Post Office Protocol 3 (POP3):

- ✓ Post Office Protocol version 3 (POP3) is a standard mail protocol used to **receive emails** from a remote server to a local email client.
- ✓ POP3 allows you to download email messages on your local computer and read them even when you are offline. Note, that when you use POP3 to connect to your email account, messages are downloaded locally and removed from the servers.
- ✓ This means that if you access your account from multiple locations, that may not be the best option for you.
- ✓ On the other hand, if you use POP3, your messages are stored on your local computer, which reduces the space your email account uses on your web server.
- ✓ By default, the POP3 protocol works on two ports:
- ✓ **Port 110** - this is the default POP3

8. Internet Message Access Protocol:

- ✓ The Internet Message Access Protocol (IMAP) is a mail protocol usedfor accessing email on a remote web server from a local client.
- ✓ IMAP and POP3 are the two most commonly used Internet mailprotocols for **retrieving emails**.
- ✓ IMAP allows the client program to manipulate the e-mail messageson the server without downloading them on the local computer.
- ✓ IMAP enables the user to search the e-mails.
- ✓ This does not involve in transfer of mails. It allows users to access the received mails.
- ✓ It enables us to take any action such as download, delete the mail without reading it.
- ✓ Both protocols are supported by all modern email clients and web servers.
- ✓ While the POP3 protocol assumes that your email is being accessed only from one application, IMAP allows simultaneous access by multiple clients. This is why IMAP is more suitable for you if you're going to access your email from different locations or if your messages are managed by multiple users.
- ✓ **Port 143** - this is the default IMAP port

1.3: WWW (WORLD WIDE WEB)

WORLD WIDE WEB (WWW): World Wide Web is a collection of software and corresponding protocols used to access the resources over the medium of internet. It is an information-sharing model that is built on top of the Internet.

- ✓ It is an information-sharing model that is built on top of theinternet.
- ✓ Also called web or www, it is a collection of information, resources, pictures, and sounds, multimedia on the internet that are linked and connected together.
- ✓ The web uses the HTTP protocol, only one of the languages spoken over the internet, to transmit data.
- ✓ Web services, which use HTTP to allow applications to communicate in order to exchange business logic, use the web to share information.
- ✓ The web also utilizes browsers, such as Internet Explorer or Firefox, to access Web documents called webpages that are linked to each other via hyperlinks. Web documents also contain graphics, sounds, text and video.

- ❖ WWW contains huge amount of documents, images and other resources which can be accessed using ***hyperlinks.*** People use internet through Web.
- ❖ Web is the largest transformable-information construct that was introduced by Tim Burners-Lee in 1989.

QUICK POINTS ABOUT THE WEB:

- ✓ It is a system of internet servers that support specially formatted documents.
- ✓ Documents are formatted in a markup language that supportslinks to other documents.
- ✓ You can jump from one document to another simply by clicking onhot spots (hyperlinks).
- ✓ Applications called web browsers that make it easy to access theWorld Wide Web.
- ✓ There are more than 1,275,000,000 websites.

Internet Vs. WWW

	Internet	WWW
Definition	Internet is a massive network of networks, a networking infrastructure	WWW is a way of accessing information over the medium of the internet
Purpose	It connects millions of computers together globally forming a network in which any computer can communicate with any other computer as long as both are connected to internet	It is an information sharing model that is built on the top of internet.
Name of the first version	ARPANET	NSFnet
Comprises	Network of computers, copper wires, fiber-optic cables 7 wireless networks	Files, folders & documents stored in various computers
Governed by	Internet Protocol	Hyper Text Transfer Protocol
Dependency	This is the base, independent of the WWW	It depends on internet to work
Creator	No such creator	Created by Tim Berners Lee in 1992.
Nature	Hardware	Software

1.4 HTTP: Hyper Text Transfer Protocol

✓ Hyper Text Transfer Protocol (HTTP) is the communication protocol used by Internet or WWW to transfer hypertext documents.

- **HTTP is based on the request-response communication model:**
  - Client sends a request
  - Server sends a response

✓ **HTTP is a stateless protocol:**
  - The protocol does not require the server to remember anything about the client between requests. In other words, the current request does not know what has been done in the previous requests.

❖ HTTP Messages:

A HTTP message is the information transaction between the client and server.

- Whenever we issue URL from our browser to get a web resource using HTTP, the browser turns the URL into a **request message** and sends it to the HTTP server.
- The HTTP server interprets the request message, and returns an appropriate **response message**, which is either the resource we request or an error message.

Two types of HTTP Message:

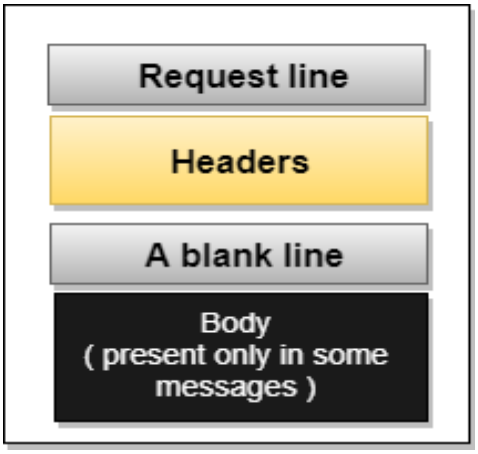
1. **HTTP Request –** Client to Server
2. **HTTP Response –** Server to Client

1. HTTP Request Message:

HTTP Requests are messages which are sent by the client or user to initiate an action on the server.

After opening a connection to the intended server, the HTTP client transmits a request in the following format

(Structure of the HTTP Request):



- Start line

- Header Field(s)
- Blank line
- Optionally, a message body

➢ The Start line is generally split into three space-separated parts;

1. HTTP Request Method
2. Request URI
3. HTTP Version

Example: A typical start line might read:

```
GET /sams/testpage.html HTTP/1.0
```

1. **HTTP Request Method**

   Common request methods:

   a. **GET**
      - Used if link is clicked or address typed in browser
      - No body in request with GET method

   b. **POST**
      - Used when submit button is clicked on a form
      - Form information contained in body of request

   c. **HEAD**
      - Requests that only header fields (no body) be returned in the response

2. **Request URI:**

   Request-URI is the portion of the requested URI that follows the host name (which is supplied by the required Host header field)
   Ex: / is Request-URI portion of http://www.example.com/

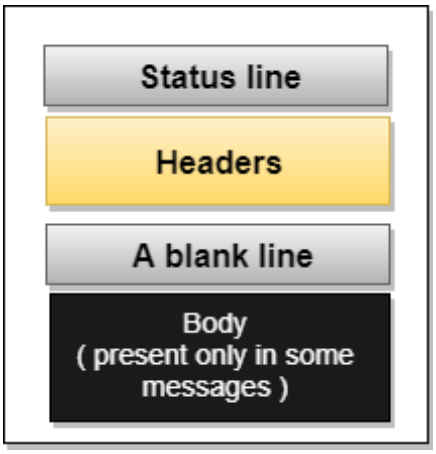
3. **HTTP Version:** HTTP 1.0 / HTTP 1.1

➢ **Request Header Fields**

- **Header field structure:**
  - *field name* : *field value*
- **Syntax**
  - Field name is not case sensitive
  - Field value may continue on multiple lines by starting continuation lines with white space
  - Field values may contain MIME types, quality values, and wildcard characters (*'s)

- **Multipurpose Internet Mail Extensions (MIME)**
  - Convention for specifying content type of a message
- In HTTP, typically used to specify content type ofbody of the response the
  - MIME content type syntax:
- *top-level type / subtype*
  - Examples: text/html, image/jpeg
- **Common header fields:**
  - **Host:** host name from URL (required)
  - **From:** Email address of user
  - **User-Agent:** type of browser sending request
  - **Accept:** MIME types of acceptable documents
  - **Accept-Language:** Languages to accept as response
  - **Accept-encoding:** Compression Methods
  - **Connection:** value ‐close‖ tells server to close connection after single request/response
  - **Content-Type:** MIME type of (POST) body, normally application/x-www-form-url encoded
  - **Content-Length:** bytes in body
  - **If-Modified-Since:** Return document only if modified sincespecified
  - **Referer:** URL of document containing link that supplied URIfor this HTTP request

Example: HTTP Request Message



2. HTTP Response Message:

HTTP Response sent by a server to the client.The response is used to provide the client with the resource it requested.

It is also used to inform the client that the action requested has been carried out.

It can also inform the client that an error occurred in processing its request.

An HTTP response contains the following things:

(Structure of the response)

- status line
- header field(s)
- blank line
- optional Message body

➢ **Status Line**

The Start line is generally split into three space-separated parts;

- HTTP version
- status code
- reason phrase (intended for human use)

1. HTTP Version: HTTP 1.0 / HTTP 1.1

2. Status Code:

- Status code is a three-digit number that indicates the result ofthe request.
- First digit is class of the status code:
  - 1=Informational
  - 2=Success
  - 3=Redirection (alternate URL is supplied)
  - 4=Client Error
  - 5=Server Error
- Other two digits provide additional information

3. Reason Phrase:

It is also known as the status text. It is a human-readable text that summarizes the meaning of the status code.

An example of the response line is as follows:

HTTP/1.1 200 OK

Here,

- HTTP/1.1 is the HTTP version.
- 200 is the status code.
- OK is the reason phrase.

Some Commonly Encountered HTTP Response Status Codes

Status Code	Explanation
200 - OK	The request succeeded.
204 - No Content	The document contains no data.
301 - Moved Permanently	The resource has permanently moved to a different URI.
401 - Not Authorized	The request needs user authentication.
403 - Forbidden	The server has refused to fulfill the request.
404 - Not Found	The requested resource does not exist on the server.
408 - Request Timeout	The client failed to send a request in the time allowed by the server.
500 - Server Error	Due to a malfunctioning script, server configuration error or similar.

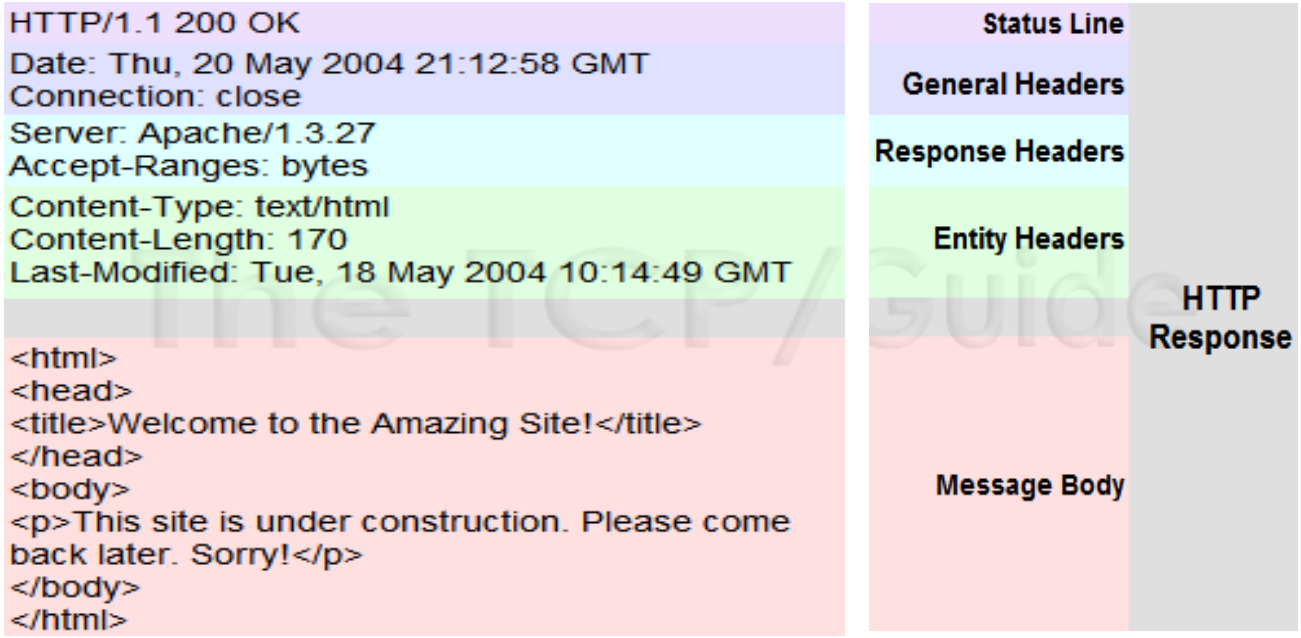
- **Response Header Fields**

The HTTP Headers for the response of the server contain the information that a client can use to find out more about the response, and about the server that sent it.

This information is used to assist the client with displaying the response to a user, with storing the response for the use of future, and with making further requests to the server now or in the future.

HTTP Response Headers

Header	Description
Server	Server software
Date	Current Date
Last-Modified	Modification date of document
Expires	Date at which document expires
Location	The location of the document in redirection responses
Pragma	A hint, e.g., no cache
MIME-version	
Link	URL of document's parent
Content-Length	Length in bytes
Allowed	Requests that user can issue, e.g., GET



Example: HTTP Response Message

1.5: WEB SERVER AND WEB CLIENT

- **WEB SERVER:**

A Web Server is basically a computer that is designed to accept requests from remote computers and send on the information requested over the internet. It is also used to host website / web application.

- The primary function of a web server is to store, process and deliverweb pages to clients.
- The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP).
- Pages delivered are most frequently HTML documents, which mayinclude images, style sheets and scripts in addition to text content.

- **Functions of a web server:**
  1. The web server accepts the requests from the browser.
  2. The user request is processed by the web server.
  3. The web server responds to the users by providing the service which they demand.
  4. The servers verify the given address for its existence, finds the necessary files, run appropriate scripts and returns back the cookies to the browser.
  5. Some servers also participate in session handling.

- **Examples:**
  - **Apache**
    - Source: The Apache Software Foundation
    - Operating System: Unix, Windows NT
    - Current Version: 2.2.6
  - **Internet Information Server**
    - Source: Microsoft
    - Operating System: Windows Server 2003
    - Current Version: 6.0
  - **Sun Java System Web Server**
    - Source: Sun Microsystems
    - Operating System: Solaris, Windows Server 2003, 2000, XP, Linux, HP-UX
    - Current Version: 7.0

➢ **WEB CLIENT:**

A Web client is the browser on the PC/Mac that makes the requests to the remote server. It is an application (e.g.

Internet Explorer, Firefox, Chrome, Safari, and Opera) running on a local device (desktop, notebook, cell phone) used to interact mainly with Web servers. A PC/Mac that uses a web (Client) browser is referred to as a Client Machine.

- The primary function of a web client is to serve content.
- A user agent, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so.

Functions of a Web Browser:

1. Reformat the URL and send a valid HTTP request.
2. When user gives the address of particular web site, the web browser converts the DNS to corresponding ip address.
3. The web browser establishes a TCP connection with the web server while processing the user's request.
4. The web browser sends the HTTP request to the web server.
5. The web browser displays the response page returned by the server in appropriate format on the client machine.

- **Examples:**
  - **Firefox**
    - Source: **Mozilla** Corporation
    - Operating System: Mac OS X, Microsoft Windows, Linux
    - Current Version:2.0.0.9
  - **Internet Explorer**
    - Source: **Microsoft**
    - Operating System: Windows, Vista
    - Current Version: 7.0
  - **Netscape**
    - Source: Netscape Corporation
    - Operating System: Mac OS X, Microsoft Windows, Linux
    - Current Version: 9.0.0.3
  - **Opera**
    - Source: **Opera Software**
    - Operating System: Microsoft Windows, Solaris, SolarisIntel, Sparc, ONX, OS/2,
    - MacOS, Linux sparc, Linux Power PC, Linux i386, Fress BSD i386, BeOS
    - Current Version: 9.24
  - **Safari**

    - Source: **Apple Inc.**
    - Operating System: Mac OS X, Microsoft Windows, Linux
    - Current Version: 3.0.4 (Beta version)

➢ **DIFFERENCE BETWEEN WEB SERVER AND WEB BROWSER:**

S.No	Web Server	Web Browser
1	Web server is essential to store all information and data of websites.	web browser are used to access and locate these information and data.
2	web server is used to make the links between websites and web browser.	Web browser is used to search something on the internet via websites.
3	web server is a program server on computer or in cloud on internet that gives the data.	Web browser is a software or application which is used for collection and presentation of data in shape of websites

➢ WEB PAGE:

A **web page** or **webpage** is a document commonly written in HyperText Markup Language (HTML) that is accessible through the Internet or other network using a browser. A web page is accessed by entering a URL address and may contain text, graphics, and hyperlinks to other web pages and files.

Two types of Web Pages:

1. Static Web Page:

A *static web page* is delivered exactly as stored, as webcontent in the web server's file system. Static pages show the same content each time they are viewed. They are written with HTML.

2. Dynamic Web Pages:

A *dynamic web page* is generated by a web application that is driven by server-side software or client-side scripting. They are written in scripting languages such as PHP, Perl, ASP, or JSP. Dynamic web pages help the browser (the client) to enhance the web page through userinput to the server.

Difference between Static and Dynamic Web Pages:

S.No	Static Web Pages	Dynamic Web Pages
1	"Static" means unchanged or constant	"Dynamic" means changing or lively.
2	They contain HTML code, which defines the structure and content of the Web page. Each time an HTML page is loaded, it looks the same. The only way the content of an HTML page will change is if the Web developer updates and publishes the file	Web pages, such as PHP, ASP, and JSP pages are dynamic Web pages. These pages contain "server-side" code, which allows the server to generate unique content each time the page is loaded. For example, the server may display the current time and date on the Web page.
3	If the file extension is ".htm" or ".html," the page is probably static	If the file extension is ".php," ".asp," or ".jsp," the page is most likely dynamic.

➢ WEB SITE:

 A Web Site is a set of interconnected webpages on WWW that includes a beginning file called homepage.

✓ Web sites are accessed by using URLs provided by the authority of the web site.

✓ Web sites are generally located on the same server, and prepared and maintained as a collection of information by a person, group, or organization.

Example: http://www.w3schools.com

➢ DIFFERENCE BETWEEN WEB SERVER AND WEB SITE:

S.No	Web Servers	Web Sites
1	The web server is a computer program which delivers content such as web sites or web pages. It responds to the request for web pages.	A web site is a set of linked web pages associated with a particular person and can be accessed via WWW.
2	Web server hosts all the web sites on WWW.	All the web sites reside on the web server.
3	Web Servers consists of hardware and software components	Web sites contain text files, images, videos and audios.

➢ WEB APPLICATION:

A **web application** or **web app** is any computer program that runs in a web browser. It is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a web browser to render the application.

- ✓ It is usable only with an active Internet connection
- ✓ It uses HTTP as its primary communications protocol.

Examples: Online ticket booking applications, Online games, google docs, etc.,

1.6: HTML 5.0

HYPERTEXT MARKUP LANGUAGE (HTML):

HTML is a markup language for describing web pages. It isscripting language for developing web pages and it tells a web browser how to format and display a web page.

- ✓ HTML was invented by Tim Berners Lee.
- ✓ It is a standard published by World Wide Web Consortium (W3C).
- ✓ HTML is derived from the Standard Generalized Markup Language (SGML) by applying SGML constructs according to a certain set of rules.
- ✓ HTML has a set of predefined tags to describe the format of a web document.
- ✓ All tags have their own syntax and attributes.

Advantages of HTML:

1. It is plain text so is easy to edit.
2. It is also fast to download.
3. Is very easy to pick up\learn.
4. It is now a standard.
5. It is supported by most browsers.
6. Simple to edit only requires a text editor.
7. Can be easily edited with WYSIWYG editors (no coding required) .
8. Can be used to present just about any kind of data.
9. Tags can be used very loosely (i.e. used to be able to omit end tags etc).
10. It is user friendly.
11. It is open technology.

Disadvantages of HTML:

1. It can create only static and plain pages so if we need dynamic pages then HTML is not useful.
2. It is static needs to be manually updated or needs some scripting support to change it in some way.
3. Need to write lot of code for making simple webpage.
4. Security features are not good in HTML.
5. If we need to write long code for making a webpage then it produces some complexity.

❖ BASIC STRUCTURE OF AN HTML DOCUMENT:

```
<!DOCTYPE html>
<html>

    <head>
        <title> title of the page </title>     Head Section
    </head>
    <body>
        Page design goes here                  Body Section
    </body>
<html>
```

- **The <!DOCTYPE> Declaration**
  - ✓ The <!DOCTYPE> declaration helps the browser to display a web page correctly.
  - ✓ There are different document types on the web.
  - ✓ To display a document correctly, the browser must know both type and version.
  - ✓ The doctype declaration is not case sensitive. All cases are acceptable:
    - <!DOCTYPE html>
    - <!DOCTYPE HTML>
    - <!doctype html>
    - <!Doctype Html>

Common Declarations

HTML5: <!DOCTYPE html>

HTML 4.01:
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
XHTML 1.0:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd

- **<html> tag:** Every HTML document starts with <html> tag. This tells browsers that it is the beginning of HTML document. This tag does not have any properties.
- **</html> tag:** This tag informs browsers that the end of the HTML document has been reached.
- An HTML page has basically two distinct logical sections:
  1. Head Section
  2. Body Section

1. Head Section:

- Head section contains the meta-information about the HTML page.
- This section is processed but not displayed on the screen.
- Head section may contain the following inside it:
  - ✓ **<title> tag** – used to assign title to the web page which appears on the title bar of the browser's window.
  - ✓ **JavaScript** codes for creating dynamic web pages.
  - ✓ **Cascading Style Sheets** codes for styling the document.

2. Body Section:

The body section contains text and other tags which are rendered on the screen.

❖ HTML CONSTRUCTS:

There are two main constructs in HTML:

1. Elements (also called tags)
2. Entities.

1. Elements:

HTML tags are **keywords** (tag names) surrounded by **angle brackets**. HTML element or tag is a signal to the browser that it should do something other than just throwing text on the screen. Tags are descriptions that are embedded directly into the informational text of the document.

Syntax: <tagname>content</tagname>

- HTML tags normally come **in pairs** like and
- The first tag in a pair is the **start tag,** the second tag is the **end tag**

- The end tag is written like the start tag, but with a **slash** before the tag name

2. Attributes:

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

Syntax:

<tagname attributename=||value||>content</tagname>

Example: <applet width=||100|| height=||100||>

3. Entities:

- ✓ Entities are character sequences that reproduce special characters on the browser screen.
- ✓ Reserved characters in HTML must be replaced with characterentities (references).
- ✓ If you use the less than (<) or greater than (>) signs in your text,the browser might mix them with tags.
- ✓ Character entities are used to display reserved characters inHTML.
- ✓ A character entity looks like this:

***&entity_name*; OR *&#entity_number*;**

To display a less than sign we must write: < or <

- ✓ The advantage of using an entity name, instead of a number, is that the name is easier to remember.
- ✓ The disadvantage is that browsers may not support all entity names, but the support for numbers is good.

Some Useful HTML Character Entities

Result	Description	Entity Name	Entity Number
	non-breakingspace		
<	less than	<	<
>	greater than	>	>
&	ampersand	&	&
¢	cent	¢	¢
£	pound	£	£
¥	yen	¥	¥

€	euro	€	€
©	copyright	©	©
®	registered trademark	®	®

Example: HTML page that prints "Welcome" message on the screen:

```
<!DOCTYPE html>
<html>
 <head>
      <title> My First Web Page</title>
 </head>
 <body>
      <center>
           <h1>Welcome to HTML 5</h1>
      </center>
 </body>
</html>
```

❖ DOCUMENT STRUCTURE TAGS:

1. <html> tag:

Type: Container tag

Function: Declares the document to be an HTML document. All document contents are defined between <html> and </html> tag pair.

Syntax: <html> ……</html>

2. <head> tag:

Type: Container tag

Function: contains the tags that compose the document head.

Syntax: <head> ……</head>

Related Tags: The tags that can be placed between the <head> and </head> are: <base>, <link>, <script>, <style>, <title>

3. <base> tag:

Type: Standalone tag

Function: Specifies the base URL of the document.

Syntax:

<base href=||base-url||> (or) <base target=||frame_name||>

Target – specifies the default frame name to which all the linked documents should be loaded.

Related Tags: None.

4. <link> tag:

Type: Standalone tag

Function: Makes a link between an external source and this html file.

Syntax: <link href=||url-of-the-linked-file|| title=||title|| rel=||forward-relationship|| rev=||reverse-relationship||>

Attributes:

Href – denotes the URL of the file to be linked.

Title – gives the link a descriptive title.

Rel – specifies the relationship of the linked file to the current file.

Rev – specifies how current file relates to the linked file.

Example:

```
<head>
    <link href=||styles.css|| rel=||stylesheet||>
</head>
```

5. **<script> tag: Type**:

container tag

Function: Contains script code such as JavaScript or VBScript referenced in the body section of the document.

Syntax:

```
<script language=||scripting-language||>
----- Script code ----
</script>
```

Attributes:

language – Scripting language used to write the script.

src – Specifies the URL of a file containing the script code, if the code is not written between <script> and </script> tags.

type – MIME type of the script code.

Example:

```
<head>
  <script src=||external.js|| type=||text/javascript||></script>
</head>
```

Related Tag: <noscript>

Type: Container tag.

Function: provides alternate content to use if script cannot be executed.

Syntax: <noscript> ……alternate content …….. </noscript>

Example:

```
<script language=||vbscript||>
    Document.write(—Hello World!||);
</script>
<noscript>
    You either have scripting turned off or your browser does not support VBScript
</noscript>
```

6. **<style> tag:**

Type: Container tag

Function: specifies or links a style information to the document.

Syntax: <style type=||MIME-type|| title=||title||>

--style information—
</style>

Example:

<style type=||text/css||>
H1{font:12pt;font-weight:bold;}
</style>

7. <title> tag:

Type: Container tag

Function: makes the enclosed text the tile of the web page. The text appears in the title bar of the browser.

Syntax: <title>--document title-- </title>

Attributes: None

Example:

<title>Welcome.html</title>

8. <body> tag:

Type: Container tag

Function: Contains all content and tags that compose the document body.

Syntax:

<body bgcolor=||background-color|| background=||image||
link=||unvisited-link-color|| alink=||active-link-color||
vlink=||visited-link-color|| text=||text-color||>

Attributes:

bgcolor	Specifies background color of the document	**alink**	Specifies the color of an active link
background	Specifies background image of the document	**vlink**	Specifies the color of a visited link
link	Specifies the color of a not yet visited link	**text**	Specifies the color of the enclosed text

Example:

<body bgcolor=||yellow|| >
<body bgcolor=||#0000FF|| >
<body background=||flower1.jpg||>

❖ FORMATING TAGS:

1. tag:

Type: Container tag
Function: Makes the enclosed text bold.
Syntax: text
Attributes: None
Example:

 Welcome

2. tag:

Type: Container tag
Function: Makes the enclosed text italic.
Syntax: text
Attributes: None
Example:

 Welcome

3. <u> tag:

Type: Container tag
Function: Makes the enclosed text underlined.
Syntax: <u> text </u>
Attributes: None
Example:

<u> Welcome </u>

4. <big> tag:

Type: Container tag
Function: Renders the text in the font size bigger than the default font size.
Syntax: <big> text </big>
Attributes: None
Example: <big> Welcome </big>

5. <small> tag:

Type: Container tag
Function: Renders the text in the font size smaller than the default font size.
Syntax: <small> text </small>
Attributes: None
Example:<small> Welcome </small>

6. **<s> / <strike> tag:**

Type: Container tag

Function: contains the text to be marked with strikethrough character.

Syntax: <s> text </s> (or) <strike> text </strike>

Attributes: None

Example:

<s> Welcome </s> (or) <strike> welcome </strike>

7. **<sub> tag:**

Type: Container tag

Function: contains the text to be subscript to the text that precedes it.

Syntax: _{text}

Attributes: None

Example:

a₁, a₂

8. **<sup> tag:**

Type: Container tag

Function: contains the text to be super script to the text that precedes it.

Syntax: ^{text}

Attributes: None

Example:

a¹, a²

9. ** tag:**

Type: Container tag

Function: contains the text whose font properties are to be modified.

Syntax: text

Example:

 Welcome

10. **
 tag:**

Type: Empty tag

Function: Inserts a line break in the document.

Syntax:

Example: Welcome

11. ** tag:**

Type: Container tag

Function: Centers all the text and other page components.

Syntax: text

Example:

  Welcome

12. ** tag:**

Type: Container tag

Function: defines *emphasized* text, with added semantic importance.

Syntax: **Text**

Example: welcome

13. **<mark> tag:**

Type: Container tag

Function: defines **marked** or highlighted text.

Syntax: <mark> **Text** </mark>

Example: <mark> welcome </mark>

14. **<del> tag:**

Type: Container tag

Function: defines **deleted** (removed) of text.

Syntax: <del> **Text** </del>

Example: <del> welcome </del>

15. **<ins> tag:**

Type: Container tag

Function: defines **inserted** (added) text.

Syntax: <ins> **Text** </ins>

Example: <ins> welcome </ins>

Example HTML code (formatting tags):

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

```
<hr color="blue">
<p><b>This text is bold</b>.</p>
<hr color="blue">
<p><strong>This text is strong</strong>.</p>
<hr color="blue">
<p><i>This text is italic</i>.</p>
<hr color="blue">
<p><u>This text is underlined</u>.</p>
<hr color="blue">
<p>This is <s> not </s> HTML.</p>
<hr color="blue">
<p><em>This text is emphasized</em>.</p>
<hr color="blue">
<h2>HTML <small>Small</small> Formatting</h2>
<hr color="blue">
<h2>HTML <big>Big</big> Formatting</h2>
<hr color="blue">
<h2>HTML <mark>Marked</mark> Formatting</h2>
<hr color="blue">
<p>My favorite color is <del>blue</del> red.</p>
<hr color="blue">
<p>My favorite <ins>color</ins> is red.</p>
<hr color="blue">
<p>This is <sub>subscripted</sub> text.</p>
<hr color="blue">
<p>This is <sup>superscripted</sup> text.</p>
<hr color="blue">
<font face=||Calibri|| size=||25pt|| color=||green||> This is different
font</font>
<hr color="blue">
<center> End of formatting tags </center>
<hr color="red" width="40%">
</body>
</html>
```

Output:

This text is normal.

This text is bold.

This text is strong.

This text is italic.

This text is underlined.

This is ~~not~~ HTML.

This text is emphasized.

HTML Small Formatting

HTML Big Formatting

HTML Marked Formatting

My favorite color is ~~blue~~ red.

My favorite color is red.

This is subscripted text.

This is superscripted text.

This is different font

End of formatting tags

---

16. <hr> tag:

Type: Standalone tag

Function: places a horizontal line on the page.

Syntax: <hr align=||alignment|| noshade size=||thickness|| width=||pixels or percentage||>

Attributes:

Align – LEFT | RIGHT | CENTER

Noshade – suppresses the shading effect and yields a solid line.

Size – controls the thickness of the line.

Width – length of the line specifies in pixels or percentage.

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1> Heading 1 </h1>
<hr color="red">
<h2> Heading 2 </h2>
<hr color="blue" width="25%" align="left">
<h3> Heading 3 </h3>
<hr color="green" size="7pt">
</body>
</html>
```

Output:

Heading 1

Heading 2

Heading 3

---

17. <h1> ……. <h6> tags (heading tags):

Type**: container tag**

Function: Inserts section heading within the document. It is used to specfy the relative importance of information.

Heading Levels:

Level 1 - h1 – largest font size <h1>

Level 6 – h6 – smallest font size <h6>

Syntax: <hn align=||LEFT | RIGHT | CENTER|| > text </hn> where n = 1,2,3,4,5,6

Attributes:

Align – LEFT | RIGHT | CENTER. It controls how heading is aligned on the page. Default alignment is LEFT.

Example:

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

Output:

Heading 1
Heading 2
Heading 3
Heading 4
Heading 5
Heading 6

1. ** tag:**

Type: Container tag
Function: denotes a paragraph.
Syntax: <p align=|| LEFT | RIGHT | CENTER||> paragraph text
Attributes:

Align – LEFT **|** RIGHT **|** CENTER. Controls the alignment of the text in the paragraph.

Example:

```
<p align=||center|| title=||Greetings||> Welcome</p>
```

2. **<pre> tag: Type**:

Container tag
Function: The PRE tag displays preformatted text in a fixed width font. The PRE element displays all white spaces and line breaks exactly as they appear inside the tags.
Syntax:
<pre cols=||columns|| WRAP> -- preformatted text -- </pre>

Attributes:

Cols – specifies the maximum number of characters that fit ina line.

WRAP – turns on wrapping, so that all lines fit inside thebrowser.

Example:

```
<!DOCTYPE html>
<html>
<body>
<p>
        Welcome
                To
                    HTML
</p>

<pre>
                Welcome
                        To
                            HTML
</pre>
</body>
</html>
```


Output:

```
Welcome To HTML
                        Welcome
                                To
                                  HTML
```

18. ** tag:**

Type: Container tag

Function: Generic tag for defining document block. Used for applying style information.

Syntax:

```
<span style=||style-information||   align=||LEFT|RIGHT|CENTER||>
                    Text
</span>
```

Example:

```
<span style=||font-weight:bold; color:red;>
        Welcome
                To
                        HTML
</span>
```

Output:

Welcome to HTML

2.6: HTML TABLES

- ✓ Tables are used for presenting data in tabular form i.e. to organizedata into rows and columns.
- ✓ The fundamental building blocks of a table are cells, which can contain either a **data element** or **a heading** for a column of data.
- ✓ Columns and rows will automatically size to contain their data.
- ✓ Individual table cells can span multiple rows or columns.
- ✓ Header and footer rows can be supplied.

HTML Table Tags:

Tag Name	Meaning	Tag Name	Meaning
table	Represents start of a table	**tbody**	Defines the table body
caption	Describes the table content	**tr**	Represents a table row
thead	Defines the header section	**th**	Represents a column header
tfoot	Defines the footer section	**td**	Defines data in the cell

- **Syntax of <table> tag:**

<table align=||LEFT|RIGHT|CENTER||
border=||thickness_in_pexels||
background=||Image_URL||
bordercolor=||color||
bgcolor=||color||
height=||pixels||
width=||pixels or %||
cols=”number_of_columns||
cellpadding=”pixels||
cellspaceing=”pixels||
frame=”outer_border_rendering**”>**
------ **</table>**

Attributes of Table Tag

Attribute	Meaning
Align	Specifies table alignment. An have values: left, right, and center
Border	Specifies the thickness of the table border in number of pixels
Background	Specifies the URL of the background image
Bordercolor	Specifies the color of the table border. color names or hexadecimal color codes are allowed
Bgcolor	Specifies the background color of the table. color names or hexadecimal color codes are allowed
Height	Specifies the height of the table in pixels
Width	Specifies the width of the table in pixels
Cols	Specifies the number of columns to be created in the table
Cellpadding	Specifies the amount of space in number of pixels between the cell border and its content
Cellspacing	Specifies the amount of space in number of pixels between table cells.
frame	Specifies whether a frame around the table should be created or not

- **Three Section of a table:**

A table has three distinct sections:

1. **head** section
2. **body** section
3. **foot** section

- ✓ **Head Section** (or header cell) is defined with a **<thead> element.** It contains header information such as column names.
- ✓ **Body Section** (or table body) is defined with a **<tbody> element.** It contains the table's primary data.
- ✓ **Foot Section** (or footer row) is defined with a **<tfoot> element.** It contains the footer information. The table foot section must be above the body section in the code, but the table foot displays at the bottom of the table.

- **th, tr and td elements:**

1. <th> Element:

- ✓ **<th> (table heading)** element is used to define columns in thehead section of a table.
- ✓ Most browsers center the text formatted by **th** element anddisplay them in bold.

2. <tr> Element:

- ✓ Each **<tr> (table row)** element defines an individual table row.
- ✓ **Synatx**:

<tr align=||LEFT|RIGHT|CENTER|| **bordercolor=**||color|| **bgcolor=**||color|| **height=**||pixels|| **width=**||pixels or %|| **valign=**"TOP|BOTTOM|MIDDLE|| >

</tr>

Attributes of tr tag:

Attribute	Meaning
Align	Specifies the horizontal alignment of the content of all cells in this row. Can have values: left, right and center. Default value is left.
Bordercolor	Specifies the border color of all cells in this row. color names or hexadecimal color codes are allowed
Bgcolor	Specifies the background color of all cells in this row. color names or hexadecimal color codes are allowed
Height	Specifies height, in pixels, of all cells in this row
Width	Specifies width, in pixels, of all cells in this row
valign	Specifies the vertical alignment of the content of all cells in this row. Can have values: top, bottom and middle. Default value is middle.

3. <td> Element:

- ✓ Each **<td> (table data)** element defines an individual data cells in each row.
- ✓ Each data cell contains individual piece of data.
- ✓ **Synatx:**

<td align=‖LEFT|RIGHT|CENTER‖ **bordercolor=**‖color‖ **bgcolor=**‖color‖ **height=**‖pixels‖ **width=**‖pixels or %‖ **valign=”**TOP|BOTTOM|MIDDLE‖ **colspan=**‖no_of_columns‖ **rowspan=**‖no_of_rows‖ >
data
</td>

Attribute	Meaning
Align	Specifies the horizontal alignment of the content of all cells in this row. Can have values: left, right and center. Default value is left.
Bordercolor	Specifies the border color of all cells in this row. color names or hexadecimal color codes are allowed
Bgcolor	Specifies the background color of all cells in this row. color names or hexadecimal color codes are allowed
Height	Specifies height, in pixels, of all cells in this row
Width	Specifies width, in pixels, of all cells in this row
valign	Specifies the vertical alignment of the content of all cells in this row. Can have values: top, bottom and middle. Default value is middle.
Colspan	Specifies the number of columns this cell must span
rowspan	Specifies the number of rows this cell must span

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Simple Table Demo</title>
</head>
<body>
<table border="1" width="20%">
 <caption><strong>Price of Fruits</strong></caption>
        <thead>
                <tr>
                        <th>Fruit</th>
                        <th>Price</th>
                </tr>
        </thead>
        <tfoot>
                <tr>
                        <th>Total</th>
                        <th>$3.75</th>
                </tr>
        </tfoot>
        <tbody>
                <tr>
                        <td>Orange</td>
                        <td>$0.50</td>
                </tr>
                <tr>
                        <td>Banana</td>
                        <td>$1.00</td>
                </tr>
                <tr>
                        <td>pineapple</td>
                        <td>$2.00</td>
                </tr>
                <tr>
                        <td>Apple</td>
```

```
                    <td>$0.25</td>
                </tr>
            </tbody>
        </table>
        </body>
        </html>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Simple Table Demo</title>
</head>
<body>
<table border="1" width="20%">
        <caption><strong>Price of Fruits</strong></caption>
<thead>
<tr>
 <th>Fruit</th>
<th>Price</th>

</tr>
</thead>
<tfoot>
<tr>
<th>Total</th>
<th>$3.75</th>
</tr>
</tfoot>
<tbody>
<tr>
<td>Orange</td>
<td>$0.50</td>

</tr>
<tr>
<td>Banana</td>
<td>$1.00</td>

</tr>
<tr>
<td>pineapple</td>
<td>$2.00</td>

</tr>
<tr>
 <td>Apple</td>
```

```
<td>$0.25</td>
</tr>
</tbody>
</table>
</body>
</html>
```

Rowspan & Colspan Attributes:

Rowspan – used to extend the row vertically.

Colspan – used to extend the column horizontally.

Name	**Year**
	Branch

Rowspan

Subjects	
TOC	**IP**

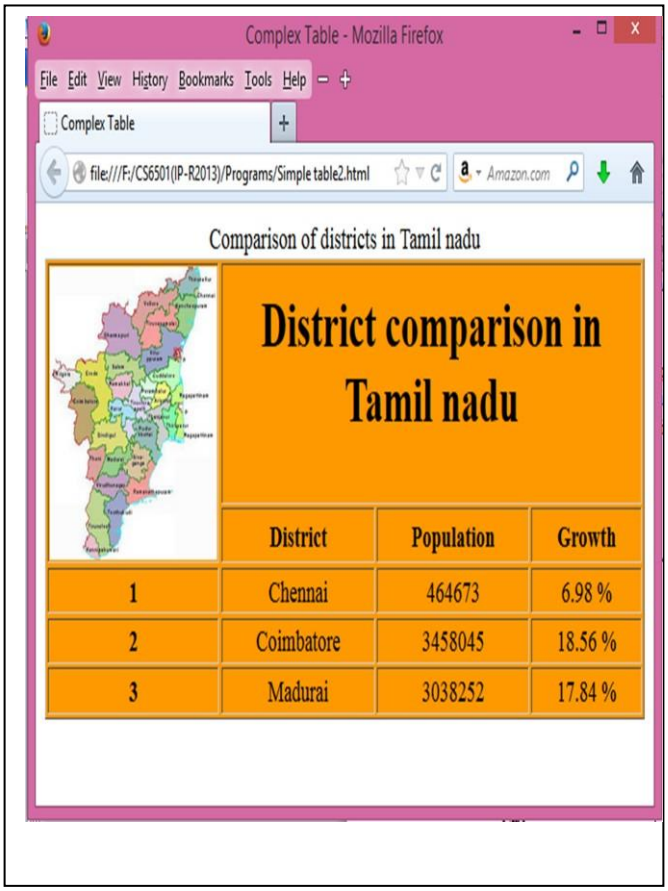
Colspan

```
<td rowspan=||2||>Name</td>
<td colspan=||2||>Subjects</td>
<td>Year</td><td>Branch</td>
<td>TOC</td><td>IP</td>
```

Example:


```
<html>
<head>
    <title>Complex Table</title>
</head>
<body>
<table border="1" bgcolor="#FF9900">
<caption> Comparison of districts in Tamil nadu</caption>
<thead>
    <tr>
    <th rowspan="2"><img src="tamil-nadu-map.jpg" width="150"
height="150"></th>
    <th colspan="3"><h1> District comparison in Tamil
nadu</h1></th>
    </tr>
    <tr>
    <th>District</th>
    <th>Population</th>
```

```
        <th>Growth</th>
        </tr>
        </thead>
        <tbody align="center">
        <tr>
                <th>1</th>
                <td>Chennai</td>
                <td>464673</td>
                <td>6.98 %</td>
        </tr>
        <tr>
                <th>2</th>
                <td>Coimbatore</td>
                <td>3458045</td>
                <td>18.56 %</td>
        </tr>
        <tr>
                <th>3</th>
                <td>Madurai</td>
                <td>3038252</td>
                <td>17.84 %</td>
        </tr>
        </tbody>
</table>
</body>
</html>
```



1.7: HTML IMAGES (<IMG> TAG)

- ✓ <img> tag is used to insert an image in the document.
- ✓ The <img> tag is empty, it contains attributes only, and does not have a closing tag
- ✓ **Syntax:**

```
<img  src=||image_url|| width=||pixels|| height=||pixels||
      alt=||text_description||
      border=||thickness_in_pixels||
      align=||TOP|MIDDLE|LEFT|RIGHT|BOTTOM
      hspace=||horizontal_space|| vspace=||vertical_space
      usemap=||map_name />
```

Where,

Src= Specifies the URL (web address) of the image.

Alt= specifies an alternate text for an image, if the image cannot be displayed. If a browser cannot find an image, it will display the alt text.

Width & Height = Optional attributes used to specify the image's width & height. If these attributes are omitted, the browser uses the image's actual width and height. Images are measured as **pixels.**

Border= used to specify the width of the border to display around the image.

Hspace & vspace= controls the amount of white space left around the image.

Usemap= used to point to an image-map. Image-map is an image with clickable areas.

Example: (To display images)

```
<!DOCTYPE html>
<html>
<body>
<center><p>Demonstration of image tag</p>
<img src="flower1.jpg" alt="Sorry! Image not found" >
<br/>
<img src="flower2.jpg" alt="Sorry! Image not found" width="128"
height="128">
```

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

1.8: LINKS (<A> TAG)

- ✓ —anchor‖ (<a>) is used for creating a clickable link called **hyperlink** between the web pages.
- ✓ **Hyperlink** references (or links to) other resources such another HTML document and images.
- ✓ When a user clicks a hyperlink, the browser tries to execute an action associated with it.
- ✓ In HTML, both text and images can act as hyperlinks.
- ✓ **Syntax:**

  ** hyperlink element **

  Where,

  Href- gives the URL of the web resource to be linked.

Target – target frame into which the linked document should be loaded.

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

HTML Links - Colors and Icons

When you move the mouse cursor over a link, two things will normally happen:

- The mouse arrow will turn into a little hand
- The color of the link element will change

By default, links will appear as 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

Hyperlinking to an E-mail Address:

- ✓Anchors can link to e-mail addresses using **mailto:URI.**
- ✓When someone click thids type of anchored link, most browser launch default e-mail program (e.g. Microsoft Outlook Express) to enable the user to write an e-mail message to the linked address.

INTERNAL LINKING:

- ✓ Internal linking is a mechanism that enables the user to jump between locations in the same document.
- ✓ Internal linking is useful for long documents that contain many sections.

- ✓ Clicking an internal linking enables users to find a section without scrolling through the entire document.

Syntax:

1. To create hyperlink: Use ―id‖ attribute in any tag to create this section as linked section.

<any-tag id=”anchor_name”> content </any-tag>

2. To point to the linked section: Use anchor name preceded with # symbol in the href attribute of <a> tag

<a href=”#anchor_name” >

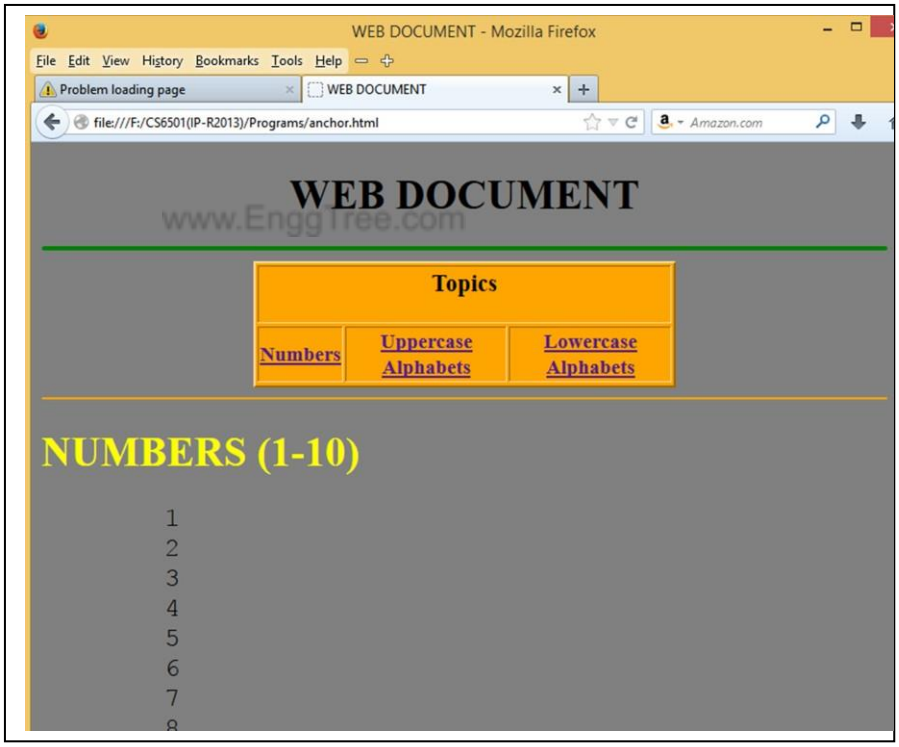
Example:

```
<HTML>
<a name="top"></a>
<HEAD><CENTER><h1>WEB DOCUMENT</h1></CENTER>
<TITLE>WEB DOCUMENT</TITLE>
</HEAD>
<hr size=4pt color="green">
<body bgcolor="gray">
<table bgcolor="orange" border=2 bordercolor="orange" width="50%"
align="center" >
<tr >
  <td colspan="3" align="center"><h3>Topics</h3></td>
</tr>
<tr>
  <th align="center" ><a href="#num">Numbers</a></th>
  <th align="center" ><a href="#upper">Uppercase
Alphabets</a></th>
  <th align="center" ><a href="#lower">Lowercase Alphabets</a></th>
</tr>
</table>
<hr color="orange">
<h1><a name="num"><font color="yellow">NUMBERS (1-
10)</font></a></h1>
<font size="5pt">
<pre>
    1
    2
```

```
        3
        4
        5
        6
        7
        8
        9
        10
</pre>
</font>
<center><a href="#top"> Go Up</a></center>
<hr color="orange">
<h1><a name="upper"><font color="yellow">UPPERCASE
ALPHABETS</font></a>
</h1>
<font size=5pt>
<pre>
                A
                B
                C
                D
                E
                F
                G
                H
                I
                J
                K
                L
                M
                N
                O
                P
                Q
                R
                S
                T
```

U
V
W
X
Y
Z
</pre>

 Go Up
<hr color="orange">
<h1>
LOWERCASE ALPHABETS</h1>
<font size=5pt>
<pre>
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w



```
            x
            y
            z
</pre>
</font>
<center><a href="#top"> Go Up</a></center>
<hr color="orange">
</body>
</html>
```

1.9: HTML LISTS

- ✓ A list is a collection of one or more items.
- ✓ HTML supports three types of lists:
  1. Unordered (bulleted) lists
  2. Ordered (numbered) lists
  3. Definition lists

1. Unordered Lists:

- ✓ An unordered list creates a bulleted list.
- ✓ An unordered list is created using the **<ul>** tag.
- ✓ Items in the list are created using the **<li>** tag and they are displayed using bullets.
- ✓ Example:

```
<ul>
        <li>C</li>
        <li>C++</li>
        <li>JAVA</li>
        <li>HTML</li>
</ul>
```

Bullet Options:

Bullet Options	Meaning	Result
	Default bullet shape is disc	• One • Two
<ul type=‖disc‖>	Disc shaped bullets are used	• One • Two
<ul type=‖circle‖>	Circle shaped bullets are used	○ One

		○ Two
<ul type=‖square‖>	Square shaped bullets are used	▪ One ▪ Two

2. Ordered Lists:

- ✓ An ordered list creates a numbered list.
- ✓ An ordered list is created using the **<ol>** tag.
- ✓ Items in the list are created using the **<li>** tag and they are displayed using bullets.
- ✓ Example:

```
<ol>
    <li>C</li>
    <li>C++</li>
    <li>JAVA</li>
    <li>HTML</li>
</ol>
```

1. C
2. C++
3. JAVA
4. HTML

Numbering Options:

Numbering options	Meaning	Result	Numbering options	Meaning	Result
	Use default number type	1. One 2. Two	<ol type=‖a‖>	Use lowercase letters for numbering	a. One b. Two
<ol type=‖1‖>	Use default number type	1. One 2. Two	<ol type=‖a‖ start=‖4‖>	Use lowercase letters stating from d	d. One e. Two
<ol type=‖A‖>	Use uppercase letter for numbering	A. One B. Two	<ol type=‖i‖>	Use small roman numbers	i. One ii. Two
<ol type=‖3‖>	Use default number	3. One 4. Two	<ol type=‖i‖ start=‖2‖>	Use small roman	ii. One iii. Two

	starting form 3			numbers starting from ii	
<ol type=‖A‖ start=‖4‖>	Use uppercase letters starting from D	D. One E. Two	<ol type=‖I‖>	Use capital roman numbers	I. One II. Two

3. Definition List:

- ✓ A definition list is the one where list items consist of two parts:
  - a term
  - its description
- ✓ Definition List is created using **<dl>** tag.
- ✓ The term part and the definition part of each item are created using **<dt>** (definition term) and **<dd>** (definition description) tags respectively.
- ✓ Example:

```
<dl>
    <dt> dl tag </dt>
    <dd> it is the outermost tag of definition list</dd>
    <dt> dt tag </dt>
    <dd> Contains the item to be described</dd>
    <dt> dd tag </dt>
    <dd> Contains description of the item</dd>
    <dd> Each term may have multiple descriptions</dd>
</dl>
```


dl tag
    it is the outermost tag of definition list
dt tag
    Contains the item to be described
dd tag
    Contains description of the item
    Each term may have multiple descriptions

- **NESTED LISTS:**
  - ✓ Lists may be nested to represent hierarchical relationships.
  - ✓ A web browser indents each nested list to indicate a hierarchical relationship.
  - ✓ Any of the three types of lists may be nested in another.
  - ✓ Arbitrary levels of nesting is possible

Example: (Three types of lists and Nested Lists):

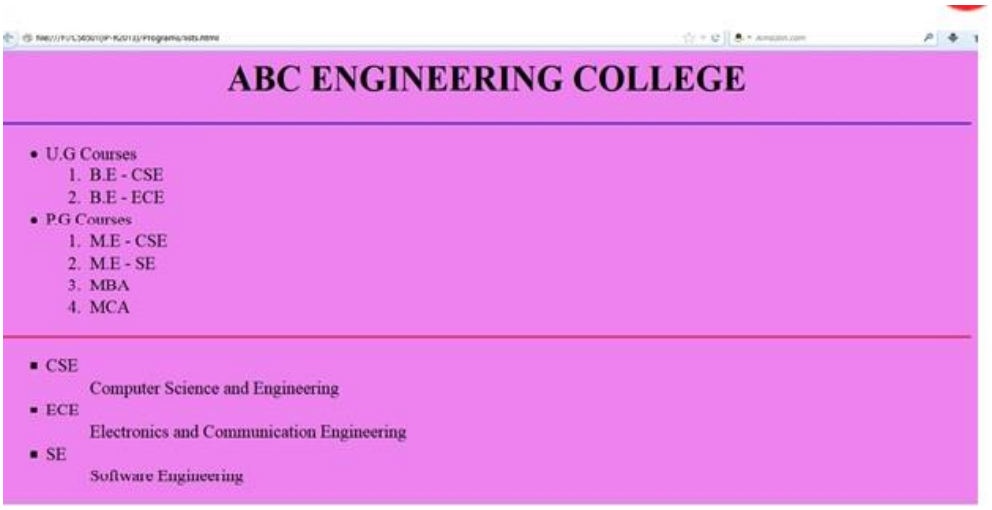
```
<!DOCTYPE html>
<html>
        <head>
         <title> List Example</title>
        </head>
   <body bgcolor="violet">
        <center>
        <h1> ABC ENGINEERING COLLEGE</h1>
        </center>
  <hr color="blue">
  <ul>
          <li> U.G Courses
                  <ol>
                        <li> B.E - CSE </li>
                  <li> B.E - ECE </li>
                  </ol>
          </li>
          <li> P.G Courses
                   <ol>
 Outer List
                  <li> M.E - CSE </li>
                  <li> M.E - SE </li>
                        <li> MBA </li>
                  <li> MCA </li>
                </ol>
               </li>
 </ul>
 <hr color="red">
```

Inner List

Inner List

```
<ul type="square">
        <li>
                <dl>
                        <dt> CSE </dt>
                        <dd> Computer Science and Engineering</dd>
                </dl>
        </li>

        <li>
                <dl>
                        <dt> ECE</dt>
                        <dd> Electronics and Communication
Engineering</dd>
                </dl>
        </li>

        <li>
                <dl>
                        <dt> SE </dt>
                        <dd> Software Engineering</dd>
                </dl>
        </li>
 </ul>
</body>
</html>
```

Output:



1.10: HTML 5 CONTROL ELEMENTS

HTML Forms

- ✓ An HTML form us used to allow a user to input data on a webpage.
- ✓ Forms acts as the visible or front-end portion of interactive webpages.
- ✓ Forms hold interface components to gather data from users. User enter information into form fields or controls and click abutton to submit the data.
- ✓ The browser then packages the data, opens an HTTP connection and sends the data to the server for processing.

Creating Forms:

Each HTML form has three main components:

[1] The form header
[2] One or more named input fields
[3] One or more action buttons

➢ **<form> ELEMENT:**

The **<form>** element is used to create a form.

Syntax:

<form name=||form_name|| **accept=**||MIME _type|| **action=**||URL_of_the_Script||
Method=||GET|POST|| **target=**||frame_name|| **>**
- **Form elements --**
</form>

Attributes:

Attribute	Description
accept-charset	Specifies the charset used in the submitted form (default: the page charset).
action	Specifies an address (url) where to submit the form (default: the submitting page).
autocomplete	Specifies if the browser should autocomplete the form (default: on).
enctype	Specifies the encoding of the submitted data (default: is url-encoded).

method	Specifies the HTTP method used when submitting the form (default: GET).
name	Specifies a name used to identify the form (for DOM usage: document.forms.name).
novalidate	Specifies that the browser should not validate the form.
target	Specifies the target of the address in the action attribute (default: _self).

Example:

```
<form name=||form1|| method=||get||
action="action_page.php">
----- form elements ----
</form>
```

- **<input> ELEMENT:**
  - ✓ The **<input>** element is the most important **form element**.
  - ✓ The <input> element has many variations, depending on the **type** attribute.

  - ✓ Here are the types used in this chapter:

Type	Description
text	Defines normal text input
radio	Defines radio button input (for selecting one of many choices)
submit	Defines a submit button (for submitting the form)

1. INPUT TYPE: textfield:

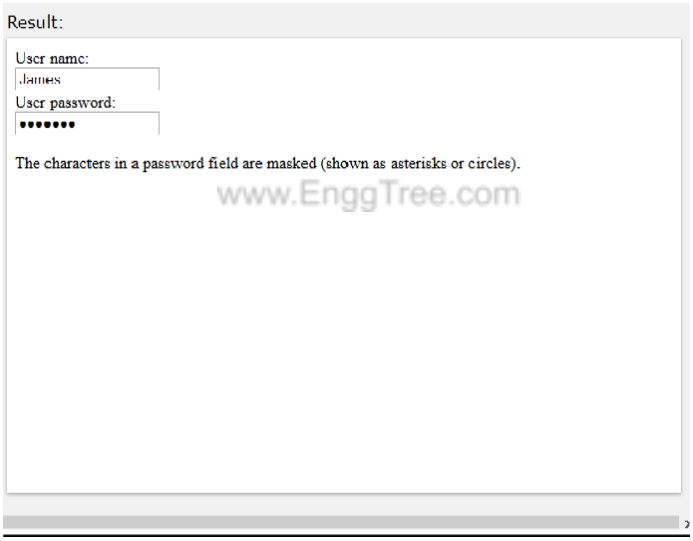
- ✓ This control simply displays a box to input a single line text.
- ✓ **<input type="text">** defines a one-line input field for **text input**.
- ✓ Default text can be included in a text field using value attribute.
- ✓ Syntax: **<input type=”text” name=”name” value=”” size=” “ />**

2. INPUT TYPE: password:

- ✓ This control simply creates an password field.
- ✓ **<input type="password">** defines a password field in the form.
- ✓ The characters in a password field are masked (shown as asterisks or circles).
- ✓ Syntax: **<input type=”password” name=”pwd” value=”” size=” “ />**

Example:

```
<!DOCTYPE html>

<html>
<body>
<form action="">
User name:<br>
<input type="text" name="userid">
<br>
User password:<br>
<input type="password" name="psw">
</form>
<p>The characters in a password field are masked (shown as asterisks or circles).</p>
</body>
</html>
```



3. INPUT TYPE: submit:

- ✓ **<input type="submit">** defines a button for **submitting** form input to a **form-handler**.
- ✓ The form-handler is typically a server page with a script for processing input data.
- ✓ The form-handler is specified in the form's action attribute:
- ✓ Syntax: **<input type="submit" name="caption" />**

Example:

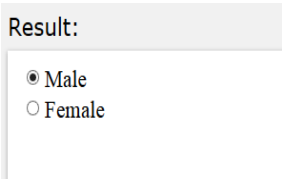
```
<form action="action_page.php">
First name:<br>
<input type="text" name="firstname" value="Mickey">
```

```
<br>
Last name:<br>
<input type="text" name="lastname" value="Mouse">
<br><br>
<input type="submit" value="Submit">
</form>
```

4. INPUT TYPE: radio:

- ✓ **<input type="radio">** defines a radio button.
- ✓ Radio buttons are used to prevent users with a set of choices from which they can choose only one of several options.
- ✓ Syntax: **<input type=”radio” name=”Name” value=”value” [CHECKED] /> option**
- ✓ **One or more radio** buttons **that all have the same value for theirname attribute forms a radio button set.**
- ✓ The browser will always display exactly one of the radio buttons as selected button.
- ✓ CHECKED attribute is used to indicate which button in a set should be checked initially.
- ✓ Example:

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

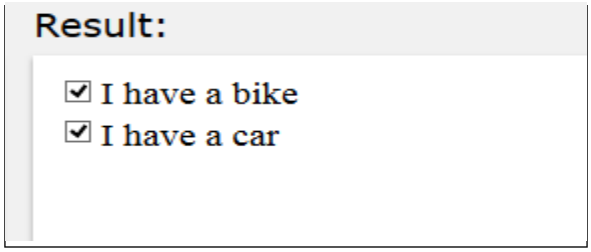



5. INPUT TYPE: checkbox:

- ✓ **<input type="checkbox">** defines a checkbox.
- ✓ Check boxes are used to provide users with several choices from which they can select as many of the choices they want.
- ✓ Syntax: **<input type=”checkbox” name=”Name” value=”value” [CHECKED] /> option**
- ✓ The information in the value attribute of the checkbox will be returned to the web server if the user checks the corresponding checkbox and submits the form.

- ✓ CHECKED attribute is used to indicate which button in a set should be checked initially.
- ✓ Example:

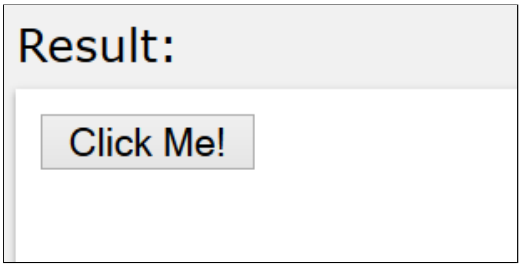
```
form>
<input type="checkbox" name="vehicle" value="Bike">I have a bike
<br>
<input type="checkbox" name="vehicle" value="Car">I have a car
</form>
```

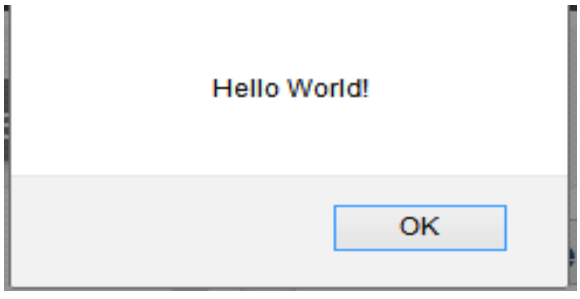


6. INPUT TYPE: button:

- ✓ **<input type="button">** creates a button.
- ✓ Syntax: **<input type="button" name="Name" value="caption" />**
- ✓ **Example:**


```
<!DOCTYPE html>
<html>
<body>
<input type="button" onclick="alert('Hello World!')" value="Click Me!">
</body>
</html>
```





➢ **TEXTAREA ELEMENT:**

- ✓ This control allows the user to enter multiple lines of data.
- ✓ It is generated on the web page using **<textarea> and </textarea>** tag pair.
- ✓ Size of the textarea is specified usingthe two attributes rows and cols.

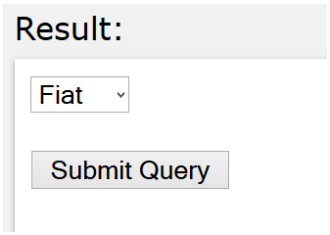
✓ Syntax: **<textarea name="text1" rows="5" cols="60">**
Please enter the text here
</textarea>

- **SELECT ELEMENT:**
  - ✓ A menu is defined using —select|| element. It defines the drop-down list.
  - ✓ It is used to produce scrollable option menus.
  - ✓ **Syntax:**

```
<select name="Name" [size="size"] [MULTIPLE] >
    <option value="value" [selected] > Option1
    </option>
    <option value="value" [selected] > Option2
    </option>
</select>
```

  - ✓ Option element is used to define the options to select.
  - ✓ [selected] attribute is used to specify which option should be selected by default (predefined option).
  - ✓ Example:

```
<form action="action_page.php">
<select name="cars">
<option value="volvo">Volvo</option>
<option value="saab">Saab</option>
<option value="fiat">Fiat</option>
<option value="audi">Audi</option>
</select>
<br><br>
<input type="submit">
</form>
```



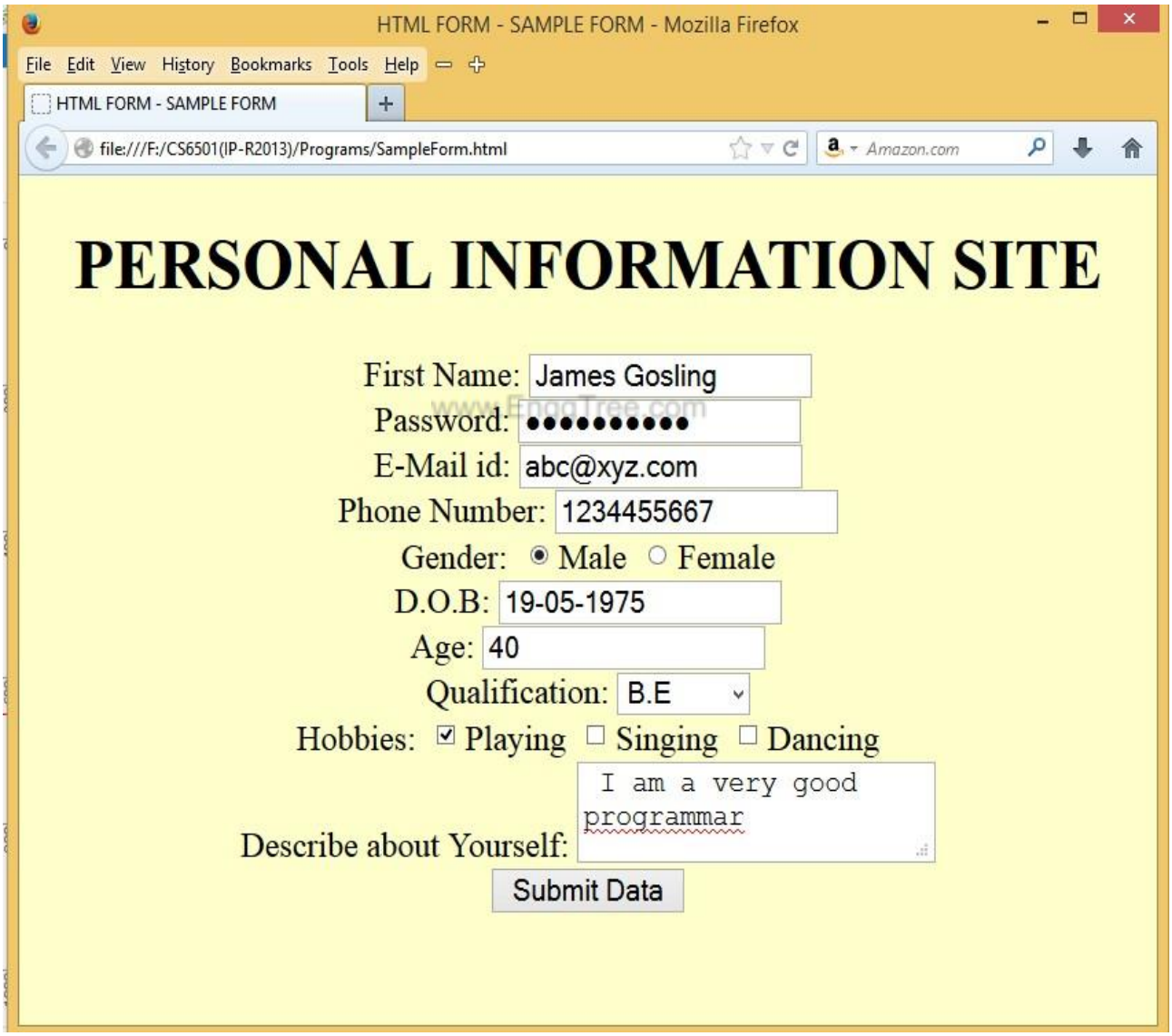
Example: Simple Form

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML FORM - SAMPLE FORM</title>
    </head>
    <body bgcolor="#FFFFCC">
    <center>
    <h1>PERSONAL INFORMATION SITE</h1>
```

```
<form method="post" action="">
        <label> First Name: <input type="text" name="uname" />
        </label>
        <br />
        <label> Password: <input type="password" name="pwd" />
        </label>
        <br />
        <label> E-Mail id: <input type="email" name="email" />
        </label>
        <br />
<label> Phone Number: <input type="number" maxlength="10"
                                name="phone" />       </label>
        <br />

        <label> Gender:
        <input type="radio" name="gender" />Male
        <input type="radio" name="gender" />Female </label><br />
        <label> D.O.B: <input type="date" name="bday" /> </label>
        <br />

        <label> Age: <input type="number" maxlength="3"
                            name="age" /> </label>
        <br />

        <label> Qualification:
        <select>
              <option value="be">B.E</option>
              <option value="ba">B.A</option>
              <option value="bcom">B.COM</option>
              <option value="bba">B.B.A</option>
        </select></label>
        <br />

        <label> Hobbies:
        <input type="checkbox" name="c1" />Playing
        <input type="checkbox" name="c2" />Singing
      <input type="checkbox" name="c3" />Dancing </label> <br />
```

```
            <label> Describe about Yourself:
            <textarea> </textarea> </label><br /> <br/>
            <input type="submit" value="Submit Data" />
        </form>
        </center>
        </body>
</html>
```

Output



➢ HTML5 FORM ELEMENTS

HTML5 added the following form elements:

- <datalist>
- <keygen>
- <output>

1. <datalist> Element:

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

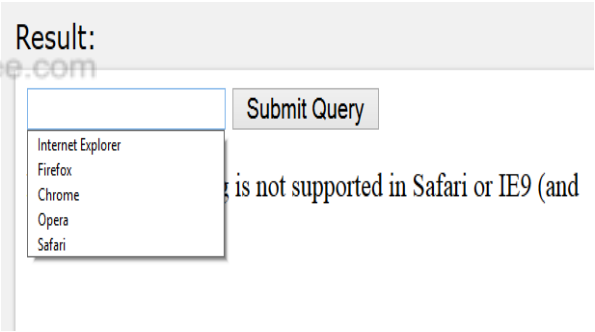
✓ Users will see a drop-down list of pre-defined options as they input data.

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

Example

An <input> element with pre-defined values in a <datalist>:

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



2. <keygen> Element:

✓ The purpose of the **<keygen>** element is to provide a secure way to authenticate users.

✓ The <keygen> element specifies a key-pair generator field in a form.

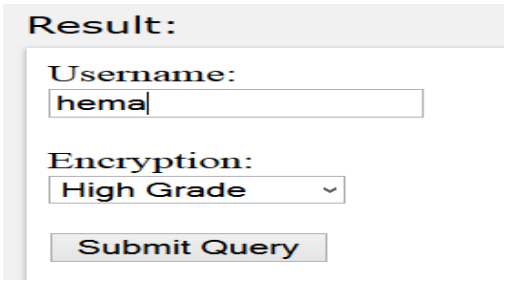
✓ When the form is submitted, two keys are generated, one private and one public.

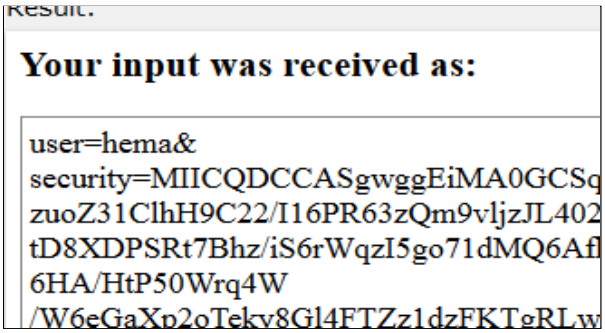
✓ The private key is stored locally, and the public key is sent to the server.

✓ The public key could be used to generate a client certificate to authenticate the user in the future.

- ✓ Example:

```
<form action="action_page.php">
Username:
<br>
<input type="text" name="user">
<br><br>
Encryption:
<br>
<keygen name="security">
<br><br>
<input type="submit"> </form>
```





3. <output> Element:

- ✓ The <output> element represents the result of a calculation (like one performed by a script).
- ✓ Example:

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

❖ HTML5 INPUT TYPES

HTML5 added several new input types:

- color
- date
- datetime
- datetime-local
- email
- month
- number
- range
- search
- tel
- time
- url
- week

1. Input Type: number

The **<input type="number">** is used for input fields that should contain a numeric value.You can set restrictions on the numbers.
Depending on browser support, the restrictions can apply to the input field.

Example

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

Input Restrictions:

Here is a list of some common input restrictions (some are new in HTML5):

Attribute	Description
disabled	Specifies that an input field should be disabled
max	Specifies the maximum value for an input field
maxlength	Specifies the maximum number of character for an input field
min	Specifies the minimum value for an input field
pattern	Specifies a regular expression to check the input value against
readonly	Specifies that an input field is read only (cannot be changed)
required	Specifies that an input field is required (must be filled out)
size	Specifies the width (in characters) of an input field
step	Specifies the legal number intervals for an input field
value	Specifies the default value for an input field

Example:

```
<form>
  Quantity:
  <input type="number" name="points" min="0" max="100" step="10"
value="30">
</form>
```


2. Input Type: date

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

Example

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

Example

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

3. Input Type: color

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

Example

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

4. Input Type: range

The **<input type="range">** is used for input fields that should contain a value within a range. Depending on browser support, the input field can be displayed as a slider control.

Example

```
<form>
  <input type="range" name="points" min="0" max="10">
</form>
```

5. Input Type: month

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

Example

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

6. Input Type: week

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

Example

```
<form>
  Select a week:
  <input type="week" name="week_year">
</form>
```

7. Input Type: time

The **<input type="time">** allows the user to select a time (no time zone). Depending on browser support, a time picker can show up in the input field.

Example

```
<form>
  Select a time:
  <input type="time" name="usr_time">
</form>
```

8. Input Type: datetime

The **<input type="datetime">** allows the user to select a date and time (with time zone).

Example

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

9. Input Type: email

The **<input type="email">** is used for input fields that should contain an e-mail address.

Depending on browser support, the e-mail address can be automatically validated when submitted.

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

Example

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

10. Input Type: search

The **<input type="search">** is used for search fields (a search field behaves like a regular text field).

Example

```
<form>
  Search Google:
  <input type="search" name="googlesearch">
</form>
```

11. Input Type: tel

The <input type="tel"> **is used for input fields that should contain a telephone number.**

The tel type is currently supported only in Safari 8.

Example

```
<form>
  Telephone:
  <input type="tel" name="usrtel">
</form>
```

12. Input Type: url

The **<input type="url">** is used for input fields that should contain a URL address.

Depending on browser support, the url field can be automatically validated when submitted

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

Example

```
<form>
  Add your homepage:
  <input type="url" name="homepage">
</form>
```


1.11: HTML 5 SEMANTIC ELEMENTS

What are Semantic Elements?

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

Examples of **non-semantic** elements: and - Tells nothing about its content.

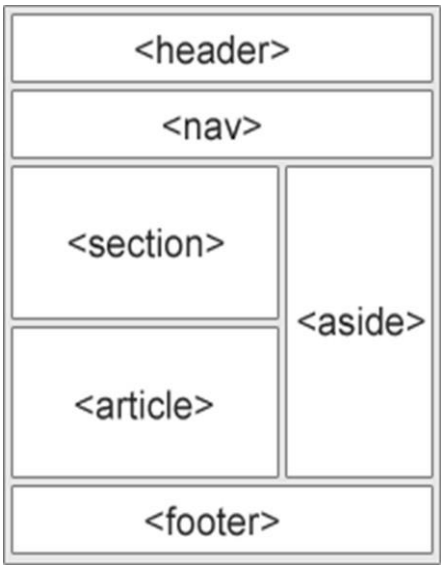
Examples of **semantic** elements: <form>, <table>, and <article> - Clearly defines its content.

Below is a list of some of the semantic elements in HTML.

Tag	Description
<article>	Defines independent, self-contained content
<aside>	Defines content aside from the page content
<details>	Defines additional details that the user can view or hide
<figcaption>	Defines a caption for a <figure> element
<figure>	Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
<footer>	Defines a footer for a document or section
<header>	Specifies a header for a document or section
<main>	Specifies the main content of a document
<mark>	Defines marked/highlighted text
<nav>	Defines navigation links
<section>	Defines a section in a document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time

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

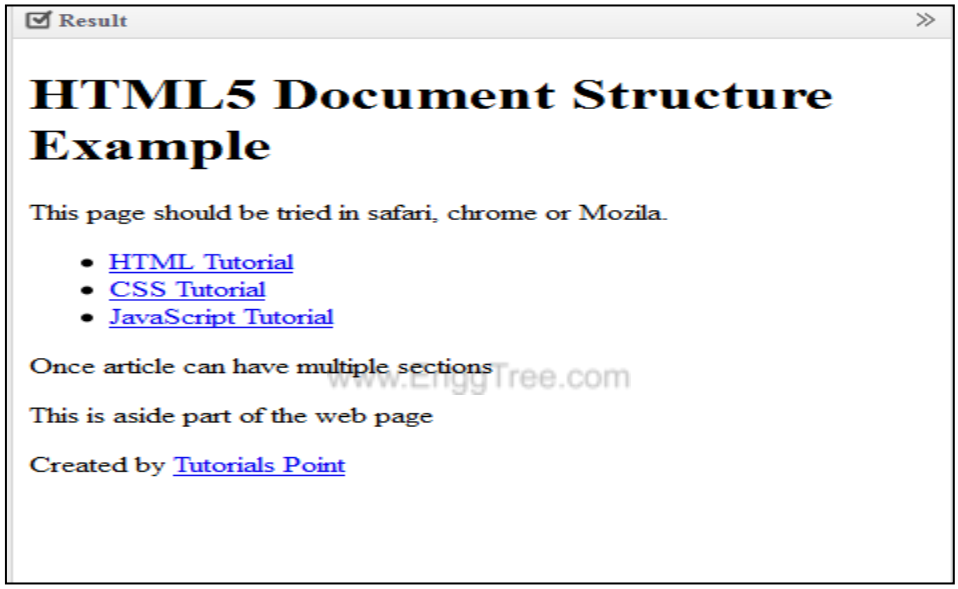
- <article>
- <aside>
- <figcaption>
- <figure>
- <footer>
- <header>
- <nav>
- <section>
- <time>



Example:

```
<!DOCTYPE html>

<html>

  <head>
    <meta charset="utf-8">
    <title>...</title>
  </head>

  <body>

    <header role="banner">
      <h1>HTML5 Document Structure Example</h1>
      <p>This page should be tried in safari, chrome or Mozila.</p>
    </header>

    <nav>

      <ul>
        <li><a href="http://www.tutorialspoint.com/html">HTML
Tutorial</a></li>
        <li><a href="http://www.tutorialspoint.com/css">CSS
Tutorial</a></li>
        <li><a
href="http://www.tutorialspoint.com/javascript">JavaScript
Tutorial</a></li>
      </ul>

    </nav>

    <article>
      <section>
        <p>Once article can have multiple sections</p>
      </section>
    </article>
```

```
    <aside>
      <p>This is aside part of the web page</p>
    </aside>

    <footer>
      <p>Created by <a href="http://tutorialspoint.com/">Tutorials
Point</a></p>
    </footer>

  </body>
</html>
```



1.12: HTML Drag and Drop API

- ✓ In HTML, any element can be dragged and dropped.
- ✓ Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

Drag and Drop Events

There are number of events which are fired during various stages of the drag and drop operation. These events are listed below –

S.No	Events & Description
1	Dragstart Fires when the user starts dragging of the object.

2	Dragenter Fired when the mouse is first moved over the target element while a drag is occurring. A listener for this event should indicate whether a drop is allowed over this location. If there are no listeners, or the listeners perform no operations, then a drop is not allowed by default.
3	Dragover This event is fired as the mouse is moved over an element when a drag is occurring. Much of the time, the operation that occurs during a listener will be the same as the dragenter event.
4	Dragleave This event is fired when the mouse leaves an element while a drag is occurring. Listeners should remove any highlighting or insertion markers used for drop feedback.
5	Drag Fires every time the mouse is moved while the object is being dragged.
6	Drop The drop event is fired on the element where the drop was occurred at the end of the drag operation. A listener would be responsible for retrieving the data being dragged and inserting it at the drop location.
7	Dragend Fires when the user releases the mouse button while dragging an object.

The DataTransfer Object

- The event listener methods for all the drag and drop events accept Event object which has a readonly attribute called dataTransfer.
- The event.dataTransfer returns DataTransfer object associated with the event as follows –

```
function EnterHandler(event) {
   DataTransfer dt = event.dataTransfer;
   ----------
}
```

- The *DataTransfer* object holds data about the drag and drop operation. This data can be retrieved and set in terms of various attributes associated with DataTransfer object as explained below –

S.No	DataTransfer attributes and their description
1	dataTransfer.dropEffect [= value] Returns the kind of operation that is currently selected. This attribute can be set, to change the selected operation. The possible values are none, copy, link, and move.
2	dataTransfer.effectAllowed [= value
	Returns the kinds of operations that are to be allowed. This attribute can be set, to change the allowed operations. The possible values are none, copy, copyLink, copyMove, link, linkMove, move, all and uninitialized.
3	dataTransfer.types
	Returns a DOMStringList listing the formats that were set in the dragstart event. In addition, if any files are being dragged, then one of the types will be the string "Files".
4	dataTransfer.clearData ([format])
	Removes the data of the specified formats. Removes all data if the argument is omitted.
5	dataTransfer.setData(format, data)
	Adds the specified data.
6	data = dataTransfer.getData(format)
	Returns the specified data. If there is no such data, returns the empty string.
7	dataTransfer.files
	Returns a FileList of the files being dragged, if any.

The example below is a simple drag and drop example:

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev) {
  ev.preventDefault();
}
function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}
function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
```

```
</script>
</head>
<body>

<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)">
</div>

<img id="drag1" src="img_logo.gif" draggable="true" ondragstart="drag(
event)" width="336" height="69">

</body>
</html>
```

❖ **Make an Element Draggable**

First of all: To make an element draggable, set the draggable attribute to true:

```
<img draggable="true">
```

❖ **What to Drag - ondragstart and setData()**

- ✓ Then, specify what should happen when the element is dragged.
- ✓ In the example above, the ondragstart attribute calls a function, drag(event), that specifies what data to be dragged.
- ✓ The dataTransfer.setData() method sets the data type and the value of the dragged data:

```
function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}
```

- ✓ In this case, the data type is "text" and the value is the id of the draggable element ("drag1").

❖ **Where to Drop - ondragover**

- ✓ The ondragover event specifies where the dragged data can be dropped.
- ✓ By default, data/elements cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element.
- ✓ This is done by calling the event.preventDefault() method for the ondragover event:

```
event.preventDefault()
```

❖ **Do the Drop - ondrop**

- ✓ When the dragged data is dropped, a drop event occurs.
- ✓ In the example above, the ondrop attribute calls a function, drop(event):

```
function drop(ev)
{
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
```

Code explained:

- Call preventDefault() to prevent the browser default handling of the data (default is open as link on drop)
- Get the dragged data with the dataTransfer.getData() method. This method will return any data that was set to the same type in the setData() method
- The dragged data is the id of the dragged element ("drag1")
- Append the dragged element into the drop element

1.13: Audio – Video controls

<audio> element:

The HTML5 <audio> element specifies a standard way to embed audio in a web page. To play an audio file in HTML, use the **<audio>** element:

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

<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>

</body>
</html>
```

- ✓ The **controls** attribute adds audio controls, like play, pause, and volume.
- ✓ Text between the <audio> and </audio> tags will display in browsers that do not support the <audio> element.
- ✓ Multiple **<source>** elements can link to different audio files. The browser will use the first recognized format.

<video> element:

The HTML5 <video> element specifies a standard way to embed a video in a web page. To show a video in HTML, use the **<video>** element:

```
<!DOCTYPE html>
<html>
<body>
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
</body>
</html>
```


Result:

- ✓ The **controls** attribute adds video controls, like play, pause, and volume.
- ✓ It is a good idea to always include **width** and **height** attributes.
- ✓ If height and width are not set, the browser does not know the size of the video. The effect will be that the page will change (or flicker) while the video loads.

- ✓ Text between the <video> and </video> tags will only display in browsers that do not support the <video> element.
- ✓ Multiple **<source>** elements can link to different video files. The browser will use the first recognized format.

HTML <video> Autoplay

To start a video automatically use the **autoplay** attribute:

Example

```
<video width="320" height="240" autoplay>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

1.14: Cascading Style Sheets

Definition:

Cascading Style Sheet (CSS) is a W3C technology that allows document authors to specify the presentation of elements on a web page (e.g fonts, spacing, colors) separately from the structure of the document (section headers, body, links). This separation ofstructure from presentation simplifies maintaining and modifying a web page.

FEATURES OF CASCADING STYLE SHEETS:

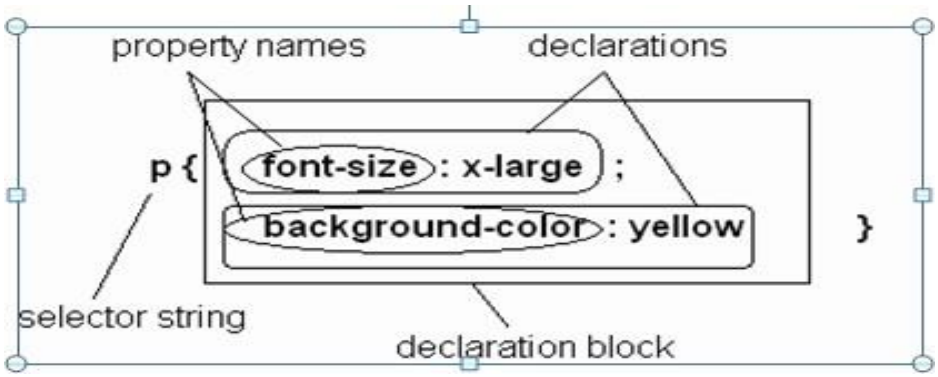
1. CSS allows separation between the information contained in the document and its presentation. Hence any change in the presentation can be made without disturbing the information of the document.
2. Style Sheets allow users to decide the style of presentation. This presentation style can be defined in a separate file. Thus the presentation can be made persistent.
3. CSS allows the developer to give the consistent appearance to all the elements of the web page.
4. CSS allows us to improve the consistent look and feel of a web site.
5. CSS provides precise control over font size, color, background color and so on.

CSS CORE SYNTAX:

A CSS style sheet consists of one or more style rules (called statements). The collection of all such rules is called **Ruleset.**

- ✓ Each ruleset consists of two parts: **1. Selector string** **2. Declaration block**
- ✓ **Declaration Block** contains a list of declarations separated by semicolon (;)
- ✓ **Selector String** indicates the elements to which the rule should apply and each declaration within the declaration block specifies a value for one style property of those elements.
- ✓ **Syntax:**

<style type="text/css">



</style>

TYPES OF SELECTOR STRING:

1. Type Selector:

- ✓ It is the simplest form of selector string which consists of the name of a single element.
- ✓ A rule can also apply to multiple elements types separating the elements by using comma (,)
- ✓ Example:

h1, p {background-color:purple}

2. Universal Selector:

- ✓ It is denoted by an asterisk (*).
- ✓ A rule denoted by universal selector can be applied for every possible element in the document.
- ✓ Example:

*** {background-color:purple; font-weight:bold}**

3. Id Selector:

- ✓ A selector preceded by a number sign (#) is called an ID selector.
- ✓ It represents an id value rather than an element type name.
- ✓ A rule defined by the ID selector can be applied to the element that matches with the id attribute value with the ID selector string.
- ✓ Example:

```
<style type="text/css">
    #myid {background-color:green}
</style>
```

```
<p id="myid">
        This is an ID selector
</p>
```

4. Class Selector:

- ✓ A selector preceded by a period or dot (.) is called an Class selector.
- ✓ It represents a generic value rather than an element type name.
- ✓ A rule defined by the class selector can be applied to the element that matches with the **class** attribute value with the Class selector string.
- ✓ Example:

```
<style type="text/css">
        .myclass {background-color:red}
</style>
<p class="myclass">
        This is an class selector
</p>
```

Note: Class selectors can also be prefixed by an element name which restricts the selector toelements of the specified type.

Example: **h1.RedText { color:red; font-size:14pt; }**

<h1 class="**RedText**"> India is my country </h1> // style applied

<h1> I am very lucky </h1> // no style applied

5. Pseudo classes:

- ✓ Using pseudo-classes we can specify special effects on the selectors.
- ✓ There are some pseudo classes which are more commonly used and those are:
  - Focus
  - Hover
  - hyperlink
- ✓ General form of pseudo class and pseudo element:

Selector:pseudo-class { declaration

}The anchor pseudo classes:

Selector	Description
a:visited	Any element with href corresponding to a URL that has been visited recently by the user.
a:link	Defined for any link that is unvisited.
a:active	Defined for any element that is being clicked.
a:hover	When the mouse is moved over the link, this pseudo class is applied.

6. At (@) Rules:

- ✓ The @import rule is used to import one style sheet file into another.
- ✓ Syntax:

@import url("filename.css");

- ✓ This rule will first read the rules form the specified file "filename.css" before continuing with the other rule in the style sheet.
- ✓ @import rule must appear at the beginning of the style sheet before any ruleset statements and it must end with semicolon (;).

Example: All types of Selector Strings

Links.css

```
a:hover {font-size:200%}
a:active {color:yellow}
a:visited {color:green}
```

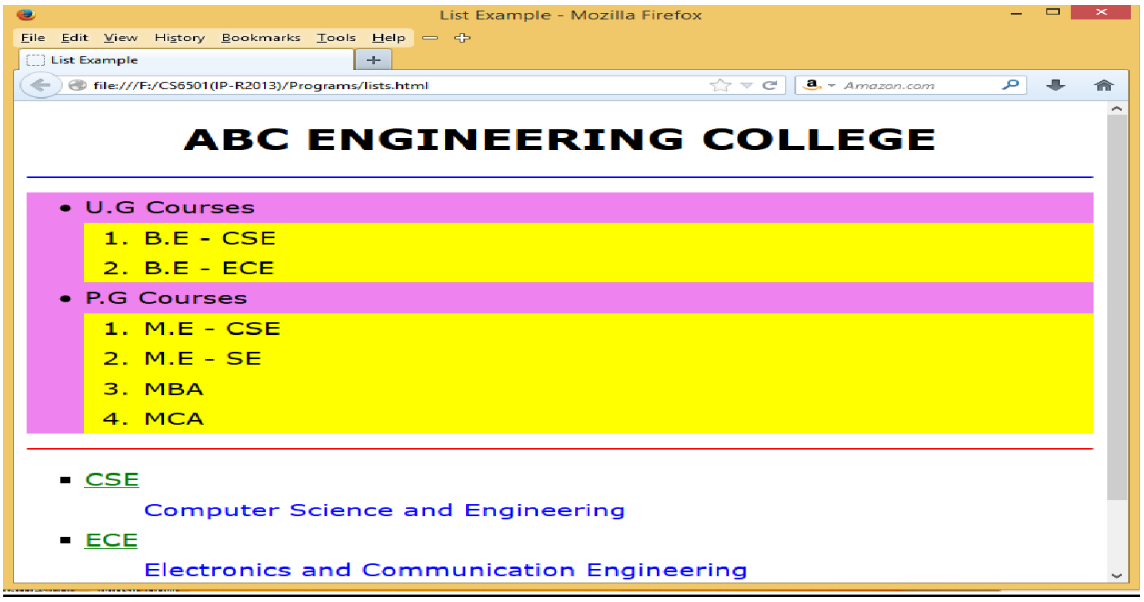
mystyle.css

```
@import url("links.css");
li { font-size: 14pt; line-height: 25pt }
* { font-family: verdana }
.unorder { background-color:yellow }
#order { background-color:violet }
dl dd {color:blue}
```


lists.html

```
<!DOCTYPE html>
<html>
  <head>
    <title> List Example</title>
        </head>
        <link href="mystyle.css" rel="stylesheet" type="text/css">
  <body >
        <center>
        <h1 color="green"> ABC ENGINEERING COLLEGE</h1>
        </center>
        <hr color="blue">
        <ul id="order">
          <li> U.G Courses
            <ol class="unorder">
                <li> B.E - CSE </li>
                <li> B.E - ECE </li>
            </ol>
```

```
</li>
<li> P.G Courses
  <ol class="unorder">
        <li> M.E - CSE </li>
        <li> M.E - SE </li>
        <li> MBA </li>
        <li> MCA </li>
  </ol>
 </li>
</ul>
<hr color="red">
<ul type="square">
<li>
 <dl>
        <dt> <a href="">CSE </a></dt>

                        <dl>

 </dl>
</li>
                        </dl>
<li>                  </li>
 <dl>                 </ul>
                </body>
               </html>
 </dl>
</li>
<li>    <dd> Software Engineering</dd>
```



1.15: TYPES OF STYLE SHEETS

To add CSS styles to our website, we can use three different ways to insert the CSS.

Different types of style sheets:

1. ***Internal / Embedded Stylesheet***
2. ***External Stylesheet***
3. ***Inline Stylesheet***

1. Internal / Embedded CSS Stylesheet:

- ✓ Internal style sheets are placed inside the <head> section of a particular web page via the style tag. **<style type="text/css">….</style>**
- ✓ These styles can be used only for the web page in which they are embedded.
- ✓ This makes it easy to apply styles like classes or id's in order to reuse the code.
- ✓ The downside of using an internal style sheet is that changes to the internal style sheet only effect the page the code is inserted into.

Example:

```
<head>
  <style type="text/css">
    h1 {color:blue;}
    h2 {color:red;}
    p {color:green;}
  </style>
</head>
```


2. External CSS Stylesheet:

- ✓ An External Stylesheet is a separate file which is then linked to the web page.
- ✓ The external style sheet is basically a text file with the extension **.css**
- ✓ So that whatever you change in the .css sheet, will effect every page in your website. This prevents you from having to make many code changes in each page. This is for "*global*" site changes.
- ✓ We must link the external style sheet to the web page to use the styles defined in that file.
- ✓ Example:

```
<head>

 <link rel="stylesheet"  type="text/css"  href="mystyle.css" />

</head>
```

Inline CSS Styles:

- ✓ The inline style uses the HTML "*style*" attribute.
- ✓ This allows CSS properties on a "*per tag*" basis.
- ✓ Inline style cannot be reused at all.
- ✓ This is not recommended, as every CSS change has to be made in every tag that has the

inline style applied to it.

- ✓ The Inline style is good for an individual CSS change that you do not use repeatedly through the site.
- ✓ The following is an example of how the inline style is used.

  ** This is a paragraph! **
- ✓ This inline style will change the color of the paragraph to red and make the font size 18 pixels.

Example:

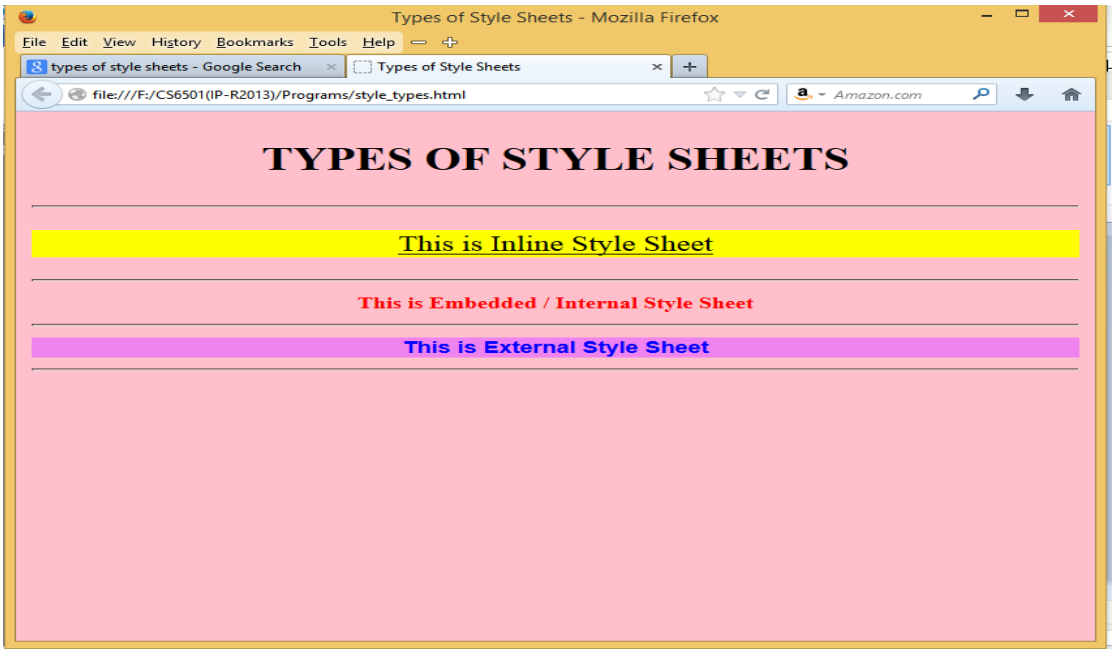
Style1.css

```
.heading
{
    font-family: Arial, Helvetica, sans-serif;
    background-color:violet;
    color:blue;
}
```

Style_types.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Types of Style Sheets</title>
        <style type="text/css">
            h1 { font-weight:bold; font-size:12px; color:red }
        </style>
        <link href="style1.css" type="text/css" rel="stylesheet">
    </head>
    <body bgcolor="pink">
        <center>
        <h2>TYPES OF STYLE SHEETS</h2>
        <hr>
        <p style="background-color:yellow;text-decoration:underline">
        This is Inline Style Sheet
        </p>
        <hr>
        <h1> This is Embedded / Internal Style Sheet</h1>
        <hr>
        <h1 class="heading">This is External Style Sheet</h1>
        <hr>
        </center>
    </body>
</html>
```

Output:



1.16: Conflicting Styles: Rule cascading and Style Inheritance

RULE CASCADING:

Consider the following declarations:

* { font-weight: bold } – this rule applies to every element of te HTML document.

#p1, #p3 { background-color:aqua } - both the rules are applied to to an element with id attribute value p3 and p1.

If multiple rules apply to an element and these rules provide declarations for different properties, then all of the declarations are applied to the element.

What will happen if multiple declarations that all apply to a single property of a single element are available?

In order to choose between multiple declarations, the browser applies **Rule Cascading**. It is a multi-stage sorting process that selects a single declaration that will supply the property value.

Stages of the sorting process:

1. Deciding which external and embedded style sheet apply to the document.
2. Associating an origin and weight with every declaration that applies to given property.

   The origin of the declaration is one of the following:

   Author: Property defined by the author of the document is chosen.

   User Agent: Default style property defined by the browser for all HTML elements.

   User: Allows users to provide a style sheet or to indicate style preferences that are treated as style rules.
3. Once the origin and weight of all declarations applying to an element property have been

4. established, they are prioritized as follows:
    1. Important declaration with user origin
    2. Important declaration with author origin
    3. Normal declaration with author origin
    4. Normal declaration with user origin
    5. Any declaration with user agent origin

5. If the top non-empty bin of the weight-origin sort contains multiple style declarations, then sort these declarations according to their **Specificity.**

1. If a declaration is part of the value of a style attribute of the element, then it is given a highest specificity value.
2. If the declaration is part of the rule set, then its specificity is determined by the selectors of the rule set.

Highest to lowest specificity for selectors:

1. ID selectors
2. class and pseudo-class selectors
3. Descendent and type selector
4. Universal selectors

5. The final step is applied if two or more declarations still have equally high weight-origin ranking and specificity even after the last sorting process.
    1. If there is a declaration in the style attribute for the element, then it is used.
    2. Otherwise, all of the style sheet rules are listed in the order in which they are processed in a top-to-bottom reading of the document.

The declaration corresponding to the rule that appear down in the list is chosen.

Example:

Style2.css

```
p {color:blue }
```

style1.css

```
@import url("style2.css");
P {color:green }
```

Mypage.html

```
<html>
<head>
  <style type="text/css">
        P {color:red }
</style>
<link rel="stylesheet" type="text/css" href="style1.css">
<style type="text/css">
        P {color: yellow}
</style>
</head>
```

In the above example, we find multiple declarations for the same property that all apply to the same element.

Apply Rule Cascading to choose the declaration. Since all the declarations are having same weight-origin and specificity, the style rule sets are ordered as follows,

p {color:red }
p {color:blue }
p {color:green }
p {color: yellow} → **This declaration is selected and the text will be displayed in yellow color.**

STYLE INHERITANCE:

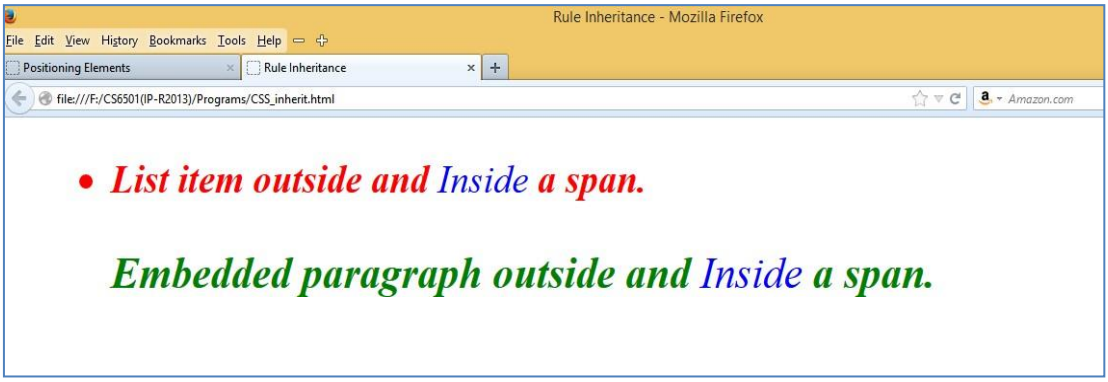
- ✓ Rule Cascading is based on the structure of the style sheets.
- ✓ Inheritance is based on the tree structure of the document.
- ✓ If a value for a style option has not been specified for an element, the element will inherit the style property form its parent.
- ✓ If the parent element does not specify the style rule, it inherits from the grand parent and so on up to the top-level element (either <html> tag or the element with style property).

Inherited style can be overridden by declaring specific style to the child.

Example:

```
<!DOCTYPE html >
<html>
 <head> <title>Rule Inheritance  </title>
 <style type="text/css">
        body { font-weight:bold; color:red }
        li {font-style:italic}
        p {font-size:larger; color:green} // overridden rule
        span {font-weight:normal; color:blue} // overridden rule
        </style>
 </head>
 <body>

        <ul>    // li inherits bold font-weight from its parent (<body>) element
                <li> List item outside and <span> Inside </span> a span.
                // p inherits italization from li
                    <p> Embedded paragraph outside and <span> Inside </span> a span. </p>
                </li>
        </ul>
        </body>
</html>
```

Output:



1.17: CSS Backgrounds

CSS background properties are used to define the background effects of an element.

CSS background properties:

Property	Description
background	Sets all the background properties in one declaration. Example: p { background: #fffff url("img") no-repeat fixed right top }
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page. Values: fixed | scroll
background-color	Sets the background color of an element. Values: With CSS, a color is most often specified by: • a HEX value - like "#ff0000" • an RGB value - like "rgb(255,0,0)" • a color name - like "red"
background-image	Sets the background image for an element. Values: None | url("")
background-position	Sets the starting position of a background image. Values: left, right, top, bottom, center
background-repeat	Sets how a background image will be repeated. Values: repeat-x | repeat-y | norepeat.

Example:

```
<!DOCTYPE html>
<html>
 <head>
        <title>CSS Backgound</title>
        <style type="text/css">

        p {
        background-color:#99FF99;
        background-attachment:fixed;
```

```
        background-image:url(flower2.jpg);
        background-position:right;
        background-repeat:repeat-y;
        height:200px }
        div {
        background:#FFFF99 url(flower1.jpg) center repeat-x scroll;
        height:150px;

        }
        </style>
 </head>
 <body>
            <center> <h1> CSS Backgorund Property</h1>
            <p> This paragraph contains a beatiful background image. </p>
            <div>ABCD EFG....  HIJK LMNOP... QRSTUVW.....UVW XYZ.</div>
            </center>
 </body>
</html>
```

1.18: CSS Borders and Border Images

❖ **CSS BORDER PROPERTIES:**

- ✓ The CSS border properties allow you to specify the style, size, and color of an element's border.
- ✓ The top, right, bottom, and left margin can be changed independently using separate properties.

Border Style: The border-style property specifies what kind of border to display.

Border-style values:

none: Defines no border

dotted: Defines a dotted border

dashed: Defines a dashed border

solid: Defines a solid border

double: Defines two borders. The width of the two borders are the same as the border-width value

groove: Defines a 3D grooved border. The effect depends on the border-color value

ridge: Defines a 3D ridged border. The effect depends on the border-color value

inset: Defines a 3D inset border. The effect depends on the border-color value

outset: Defines a 3D outset border. The effect depends on the border-color value

Border Width: The border-width property is used to set the width of the border.

The width is set in pixels, or by using one of the three pre-defined values: thin, medium, or thick.

Border Color:

The border-color property is used to set the color of the border. The color can be set by:

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

The border-style property can have from one to four values.

- **border-style: dotted solid double dashed;**
  - top border is dotted
  - right border is solid
  - bottom border is double
  - left border is dashed

- **border-style: dotted solid double;**
  - top border is dotted
  - right and left borders are solid
  - bottom border is double

- **border-style: dotted solid;**
  - top and bottom borders are dotted
  - right and left borders are solid

- **border-style: dotted;**
  - all four borders are dotted

Border - Shorthand property:

To shorten the code, it is also possible to specify all the individual border properties in one property. This is called a shorthand property.

The **border** property is shorthand for the following individual border properties:

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

Example:

```
p {
  border: 5px solid red;
}
```

CSS Rounded Borders

The border-radius property is used to add rounded borders to an element:

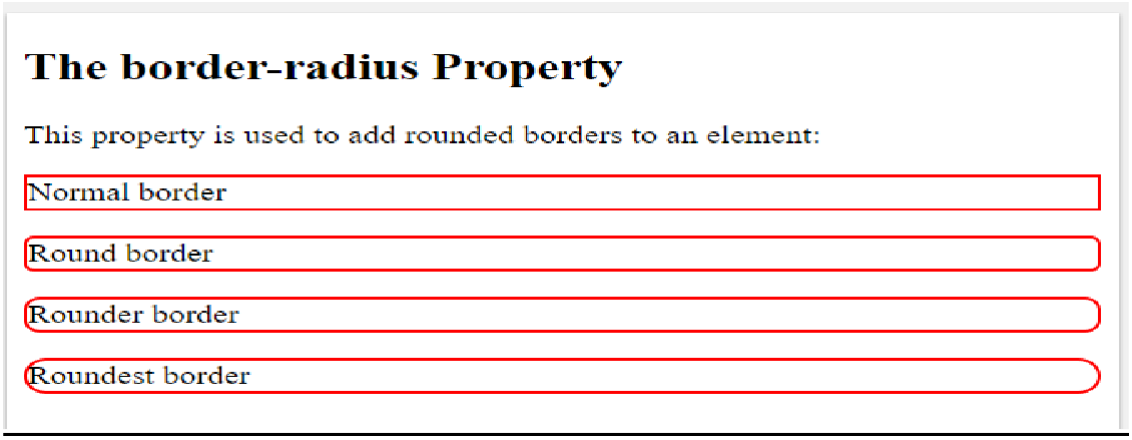
Example [CSS Border Properties]:

```
<!DOCTYPE html>
<html>
<head>
<style>
p.normal {
  border: 2px solid red;
}

p.round1 {
  border: 2px solid red;
  border-radius: 5px;
}

p.round2 {
  border: 2px solid red;
  border-radius: 8px;
}
```



```
p.round3 {
  border: 2px solid red;
  border-radius: 12px;
}
</style>
</head>
<body>

<h2>The border-radius Property</h2>
<p>This property is used to add rounded borders to an element:</p>

<p class="normal">Normal border</p>
<p class="round1">Round border</p>
<p class="round2">Rounder border</p>
<p class="round3">Roundest border</p>
```

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



❖ CSS BORDER IMAGES:

The CSS border-image property allows us to specify an image to be used instead of the normal border around an element.

The property has three parts:

1. The image to use as the border
2. Where to slice the image
3. Define whether the middle sections should be repeated or stretched

- ✓ The border-image property takes the image and slices it into nine sections, like a tic-tac-toe board. It then places the corners at the corners, and the middle sections are repeated or stretched as you specify.
- ✓ Note: For border-image to work, the element also needs the border property set!
- ✓ **Example:**

```
<style>
    #borderimg
    {
      border: 10px solid transparent;
      padding: 15px;
      border-image: url(border.png) 30 round;
    }
</style>
```

- ✓ Tip: The border-image property is actually a shorthand property for the following properties:
  - border-image-source,
  - border-image-slice,
  - border-image-width,

- border-image-outset and
- border-image-repeat

Border Image Properties

Property	Description
border-image	A shorthand property for setting all the border-image-* properties
border-image-source	Specifies the path to the image to be used as a border
border-image-slice	Specifies how to slice the border image
border-image-width	Specifies the widths of the border image
border-image-outset	Specifies the amount by which the border image area extends beyond the border box
border-image-repeat	Specifies whether the border image should be repeated, rounded or stretched

Example:


```
<!DOCTYPE html>
<html>
<head>
<style>
#borderimg1 {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(grid.png) 20% repeat;
  background-color:yellow
}

#borderimg2 {
  border: 50px solid transparent;
  padding: 15px;
  border-image: url(rose.jpg) 20% repeat;
  color:blue;
  font-size:25px
}

#borderimg3 {
```

```
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(rose.jpg) 30% round;
}
</style>
</head>
<body>

<h1>The border-image Property</h1>

<p id="borderimg1">This is the yellow background with Border Image</p>
<p id="borderimg2">border-image: url(rose.jpg) 20% round;</p>
<p id="borderimg3">border-image: url(rose.jpg) 30% round;</p>

<p><strong>Note:</strong> Internet Explorer 10, and earlier versions, do

not support the border-image property.</p>

</body>
</html>
```

1.19: CSS Colors

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

✓ **CSS Color Names**

In CSS, a color can be specified by using a predefined color name:

1. Tomato
2. Orange
3. DodgerBlue
4. MediumSeaGreen
5. Gray
6. SlateBlue
7. Violet
8. LightGray

➢ **Setting Background Color:**

Use **background-color** property to set the background color for any element

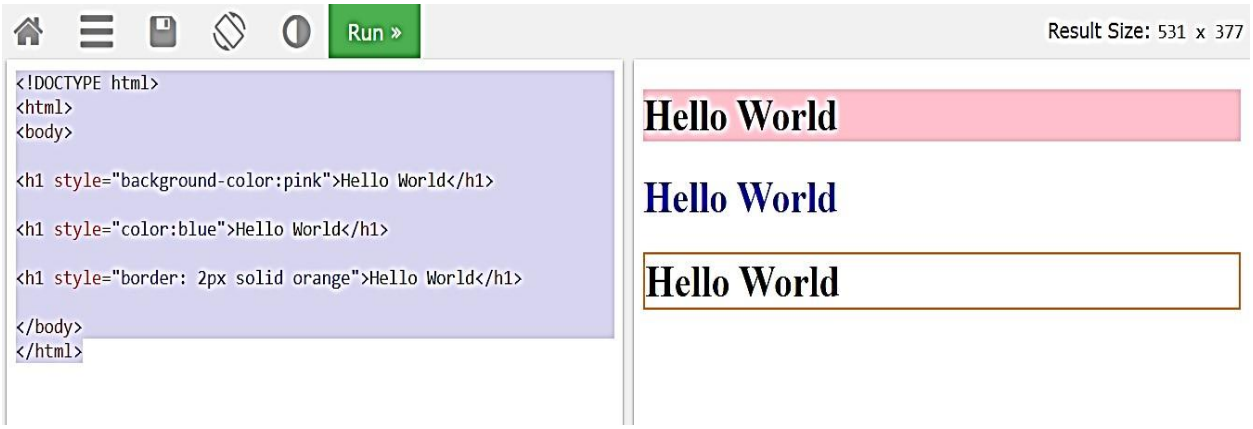
➢ **Setting Text Color:**

Use **color** property to set the color of element's text.

➢ **Setting Border Color:**

Use **border-color** property to set the border color of an element.

Example:



CSS Color Values

1) RGB Colors

✓ An RGB color value represents RED, GREEN, and BLUE light sources.

✓ **RGB Value**

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

  rgb(red, green, blue)
- Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255.
- For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255) and the others are set to 0.
- To display black, set all color parameters to 0, like this: rgb(0, 0, 0).
- To display white, set all color parameters to 255, like this: rgb(255, 255, 255).

2) HEX Colors

✓ A hexadecimal color is specified with: #RRGGBB, where the RR (red), GG (green) and BB (blue) hexadecimal integers specify the components of the color.

✓ **HEX Value**

- In CSS, a color can be specified using a hexadecimal value in the form:

  #rrggbb
- Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).
- For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).

3) HSL Colors

✓ HSL stands for hue, saturation, and lightness.

✓ **HSL Value**

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

  hsl(*hue, saturation, lightness*)
- Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
- Saturation is a percentage value, 0% means a shade of gray, and 100% is the full color.
- Lightness is also a percentage, 0% is black, 50% is neither light or dark, 100% is white

✓ **Saturation**

- Saturation can be described as the intensity of a color.
- 100% is pure color, no shades of gray
- 50% is 50% gray, but you can still see the color.
- 0% is completely gray, you can no longer see the color.

✓ **Lightness**

- The lightness of a color can be described as how much light you want to give the color, where 0% means no light (black), 50% means 50% light (neither dark nor light) 100% means full lightness (white).

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

<p>Colors with RGB, HEX and HSL Values</p>

<h1 style="background-color:rgb(255, 99, 255);">rgb(255,99,255)</h1>
<h1 style="background-color:#ffff47;">#ffff47</h1>
<h1 style="background-color:hsl(120, 50%, 50%);">hsl(120,50%, 50%)</h1>

</body>
</html>
```

Colors with RGB, HEX and HSL Values

rgb(255,99,255)

#ffff47

hsl(120,50%, 50%)

1.20: CSS Shadows

With CSS you can add shadow to text and to elements with the help of the following properties:

1. text-shadow
2. box-shadow

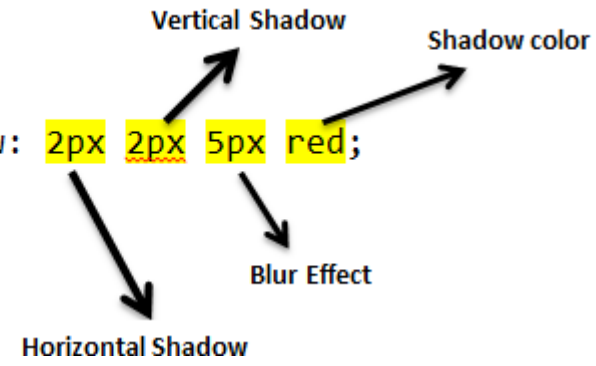
1. Text Shadow

✓ The CSS text-shadow property applies shadow to text.

✓ In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px).

✓ **Example:**

```
h1 {
  text-shadow: 2px 2px 5px red;
}
```



```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
  text-shadow: 20px 20px 3px #FF0000;
}
</style>
</head>
<body>

<h1>Text-shadow effect!</h1>

</body>
</html>
```

Text-shadow effect!

Multiple Shadows

To add more than one shadow to the text, you can add a comma-separated list of shadows.

The following example shows a red and blue neon glow shadow:

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
  text-shadow: 0 0 3px #FF0000, 0 0 5px #0000FF;
}
</style>
</head>
<body>

<h1>Text-shadow effect!</h1>

</body>
</html>
```


Text-shadow effect!

2. box-shadow

- ✓ The CSS box-shadow property applies shadow to elements.
- ✓ In its simplest use, you only specify the horizontal shadow and the vertical shadow.

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 300px;
  height: 100px;
  padding: 15px;
  background-color: yellow;
  box-shadow: 10px 10px 5px red;
}
</style>
</head>
<body>

<div>This is a div element with a box-shadow</div>

</body>
</html>
```

This is a div element with a box-shadow

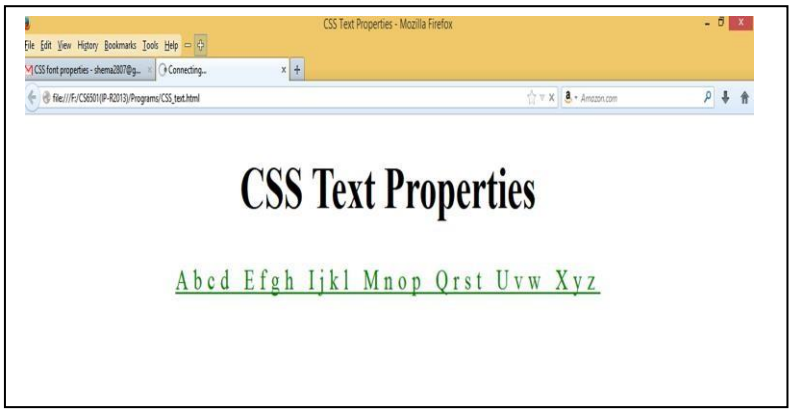
1.21: CSS Text and Font Properties

ALL CSS TEXT PROPERTIES:

Property	Description
color	Sets the color of text. Values: HEX value (#ff0000), RGB value (rgb(255,0,0)), color name (“red”)
direction	Specifies the text direction/writing direction. Values: ltr | rtl
letter-spacing	Increases or decreases the space between characters in a text
line-height	Sets the line height
text-align	Specifies the horizontal alignment of text. Values: center, justify
text-decoration	Specifies the decoration added to text. Values: none, overline, underline, line-through
text-indent	Specifies the indentation of the first line in a text-block. Value: point in pixel
text-shadow	Specifies the shadow effect added to text
text-transform	Controls the capitalization of text. Values: uppercase, lowercase, capitalize
vertical-align	Sets the vertical alignment of an element
white-space	Specifies how white-space inside an element is handled
word-spacing	Increases or decreases the space between words in a text

Example:

```
<!DOCTYPE html>
<html>
    <head>
    <title> CSS Text Properties</title>
    <style type="text/css">

    p.test1 {
    color:green;
    text-align:center;
    text-decoration:underline;
    text-indent::10pt;
    text-shadow:red;
    text-transform:capitalize;
    direction:rtl;
    letter-spacing:5px;
    }
```



```
        </style>
        </head>
        <body>
        <center><h1>CSS Text Properties</h1></center>
        <p class="test1">abcd efgh ijkl mnop qrst uvw xyz</p>
        </body>
</html>
```

CSS3 TEXT PROPERTIES:

CSS3 contains several new text features.

- text-overflow
- word-wrap
- word-break

Text Overflow:

The CSS3 text-overflow property specifies how overflowed content that is not displayed should be signaled to the user.

Word Wrapping:

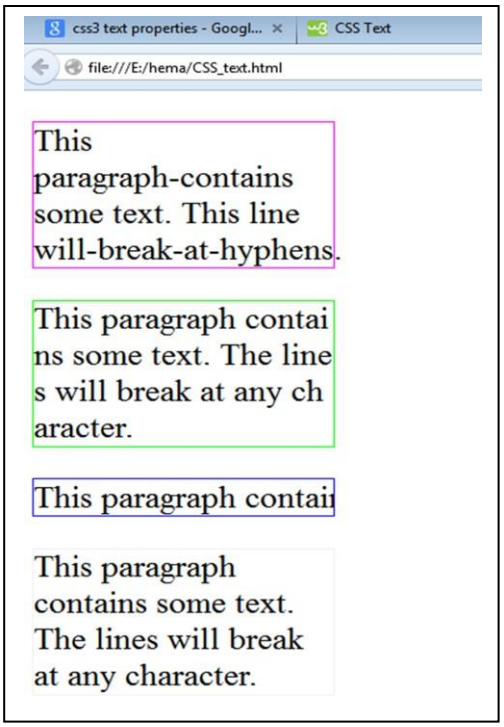
The CSS3 word-wrap property allows long words to be able to be broken and wrap onto the next line. If a word is too long to fit within an area, it expands outside.

Word Breaking:

The CSS3 word-break property specifies line breaking rules.

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
p.test1 {
    width: 140px;
    border: 1px solid #ff00ff;
    word-break: keep-all;

}

p.test2 {
    width: 140px;
    border: 1px solid #00ff00;
    word-break: break-all;
```



```
}
p.test3 {
   white-space:nowrap;
   width: 140px;
   border: 1px solid #0000ff;
   overflow:hidden;
   text-overflow:clip;
}
p.test4 {

   width: 140px;
   border: 1px solid #f0f0f0;
   word-wrap:break-word;
}
</style>
</head>
<body>

<p class="test1">This paragraph-contains some text. This line will-break-at-hyphens.</p>
<p class="test2">This paragraph contains some text. The lines will break at any character.</p>
<p class="test3">This paragraph contains some text. The lines will be clipped.</p>
<p class="test4">This paragraph contains some text. The lines will break at any character.</p>
</p>

</body>
</html>
```


CSS FONT PROPERTIES:

CSS Font Families:

In CSS, there are two types of font family names:

- **generic family** - a group of font families with a similar look (like "Serif" or "Monospace")
- **font family** - a specific font family (like "Times New Roman" or "Arial")

Generic family	Font family	Description
Serif	Times New Roman Georgia	Serif fonts have small lines at the ends on some characters
Sans-serif	Arial Verdana	"Sans" means without - these fonts do not have the lines at the ends of characters

Monospace	Courier New Lucida Console	All monospace characters have the same width

All CSS Font Properties:

Property	Description
font	Sets all the font properties in one declaration. Example: p {font: italic bold 12pt "Helvetica",Sans-serif}

font-family	Specifies the font family for text.
	Example:
	p {
	font-family: "Times New Roman", Times, serif;
	}

font-size	Sets the size of the font.
	Absolute size:
	• Sets the text to a specified size
	• Does not allow a user to change the text size in all browsers (bad for accessibility reasons)
	• Absolute size is useful when the physical size of the output is known
	Relative size:
	• Sets the size relative to surrounding elements
	• Allows a user to change the text size in browsers
	h1 {
	font-size: 40px;
	}

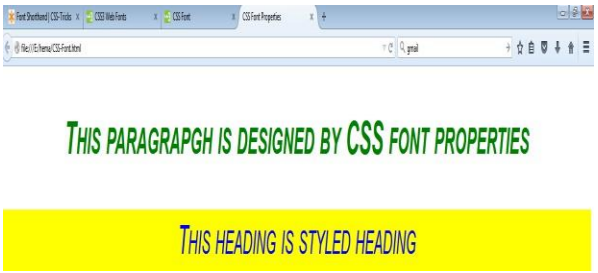
font-style	Specifies the font style for text.
	Values:
	• normal - The text is shown normally
	• italic - The text is shown in italics
	• oblique - The text is "leaning" (oblique is very similar to italic, but less supported)
	• Example:
	p.italic {
	font-style: italic;
	}

font-variant	Specifies whether or not a text should be displayed in a small-caps font.
font-weight	Specifies the weight of a font.

Example:

<html>

```
<head>
        <title>CSS Font Properties</title>
        <style type="text/css">

        p {
        font-size:50px;
        font-variant:small-caps;
        font-family:Arial;
        font-weight:bold;
        font-style:italic;
        color:green;
        }

        h1 { font:italic small-caps normal 45px/150% Arial, Helvetica, sans-serif;
            color:blue;
         background-color:yellow;
          }
        </style>
</head>
<body>
        <center> <p> This paragraph is designed by CSS font properties</p>
                <h1> This heading is styled heading</h1>
        </center>
</body>
</html>
```

CSS3 Web Fonts - The @font-face Rule:

- ✓ Web fonts allow Web designers to use fonts that are not installed on the user's computer.
- ✓ When you have found/bought the font you wish to use, just include the font file on your web server, and it will be automatically downloaded to the user when needed.
- ✓ Your "own" fonts are defined within the CSS3 @font-face rule.
- ✓ Different Font Formats:
    - **TrueType Fonts (TTF)**

- **OpenType Fonts (OTF)**
- **The Web Open Font Format (WOFF)**
- **The Web Open Font Format (WOFF 2.0)**
- **SVG Fonts/Shapes**
- **Embedded OpenType Fonts (EOT)**

✓ In the CSS3 @font-face rule you must first define a name for the font (e.g. myFirstFont), and then point to the font file.

✓ To use the font for an HTML element, refer to the name of the font (myFirstFont) through the font-family property:

Example:

```
@font-face {
   font-family:  myFirstFont;
   src: url(sansation_light.woff);
}

div {
   font-family: myFirstFont;
}
```

```
<div>
This is an example for web fonts
</div>
```

Result:

this is an example for web fonts

1.22: CSS Transformation

✓ CSS3 supports **transform property**. This transform property facilitates you to translate, rotate, scale, and skews elements.

✓ Transformation is an effect that is used to change shape, size and position.

✓ There are two type of transformation i.e. 2D and 3D transformation supported in CSS3.

CSS Transform Properties

The following table lists all the 2D transform properties:

Property	Description
transform	Applies a 2D or 3D transformation to an element
transform-origin	Allows you to change the position on transformed elements. It allows you to specify the location origin of the transform. By default, the origin is in the center of the element. For example, if you are using the transform: rotate property but want it to

rotate not from the center, but from the top left corner, you'd use the value 0% 0% or left top. For the bottom right corner, you would use 0% 100% or right bottom, etc.

Example:

```
div {
  transform-origin: left top;
  transition: transform 1s;
}

div:hover {
  transform: rotate(720deg);
}
```

CSS 2D Transforms

The CSS 2D transforms are used to re-change the structure of the element as translate, rotate, scale and skew etc.

Following is a list of 2D transforms methods:

Function	Description
matrix(*n,n,n,n,n,n*)	Defines a 2D transformation, using a matrix of six values
translate(*x,y*)	Defines a 2D translation, moving the element along the X- and the Y-axis
translateX(*n*)	Defines a 2D translation, moving the element along the X-axis
translateY(*n*)	Defines a 2D translation, moving the element along the Y-axis
scale(*x,y*)	Defines a 2D scale transformation, changing the elements width and height
scaleX(*n*)	Defines a 2D scale transformation, changing the element's width
scaleY(*n*)	Defines a 2D scale transformation, changing the element's height
rotate(*angle*)	Defines a 2D rotation, the angle is specified in the parameter
skew(*x-angle,y-angle*)	Defines a 2D skew transformation along the X- and the Y-axis
skewX(*angle*)	Defines a 2D skew transformation along the X-axis
skewY(*angle*)	Defines a 2D skew transformation along the Y-axis

1. Scale

- ✓ **The scale value allows you to increase or decrease the size of an element.**
- ✓ For example, the value 2 would transform the size to be 2 times its original size. The value 0.5 would transform the size to be half its original size.
- ✓ You can scale an element by setting parameters for the width (X-axis) or height (Y-axis). For example, transform: scaleX(2).
- ✓ Or, use the scale() shorthand to scale both axes at the same time: transform: scale(2);. Or define them independently of each other: transform: scale(2, 4);

```
<!DOCTYPE html>
<html>
<head>
<style>
        .square {
          background: darkturquoise;
          border-radius: 5px;
          height: 100px;
          margin: 100px;
          width: 100px;
        }
        .square:hover {
          transform: scale(2);
        }
</style>
</head>
<body>
<div class="square">
</div>
</body>
</html>
```


2. rotate

- ✓ **With the rotate value, the element rotates clockwise or counterclockwise by a specified number of degrees.**
- ✓ A positive value, such as 90deg, rotates the element clockwise, while a negative value, such as -90deg, rotates it counterclockwise.
- ✓ You can rotate more than a full rotation with numbers over than 360, such as 1080deg, for three full rotations.

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

```
<style>
        .square {
          background: red;
          transition: all 3s;
          height: 100px;
          margin: 100px;
          width: 100px;
        }
        .square:hover{
        transform: rotate(180deg);
        }
</style>
</head>
<body>
<div class="square">
</div>
</body>
</html>
```

3. Translate

- ✓ **The translate value moves an element left/right and up/down. The movement is based on the parameters given for the X (horizontal) Y (vertical) axes.**
- ✓ A positive X value moves the element to the right, while a negative X moves the element to the left. A positive Y value moves the element downwards and a negative Y value, upwards.
- ✓ In this example, the element will move 20 pixels to the right and 20 pixels down.


```
<!DOCTYPE html>
<html>
<head>
<style>
        .square2 {
          background: #FA54DE;
          transition: 2s;
          height: 100px;
         margin: 100px;
         width: 100px;
         position:absolute;
```

```
        }

        .square2:hover{
        transform: translate(100px,100px);
        }
</style>
</head>
<body>
<div class="square2">
</div>
</body>
</html>
```

4. Skew

- ✓ **With the skew value, the element skews (or tilts) one direction or the other based on the values given for the X and Y axes.**
- ✓ A positive X value tilts the element left, while a negative X value tilts it right.
- ✓ A positive Y value tilts the element down, and a negative Y value tilts is up. Or use a shorthand to include both X and Y properties:

CSS syntax examples for `skew`


```
div {
  transform: skewX(25deg);
  transform: skewY(10deg);
  transform: skew(25deg, 10deg);
}
div {
  transition: transform 1s;
}

div:hover {
  transform: skewX(-20px);
}
```

5. Matrix

- ✓ The matrix() method combines all the 2D transform methods into one.
- ✓ The matrix() method take six parameters, containing mathematic functions, which allows you to rotate, scale, move (translate), and skew elements.
- ✓ The parameters are as follow:
  matrix(scaleX(),skewY(),skewX(),scaleY(),translateX(),translateY())

Example

```
div {
  transform: matrix(1,2,3,4,5,6);
}
```

CSS 3D Transforms

CSS also supports 3D transformations.

Transform Properties

The following table lists all the 3D transform properties:

Property	Description
transform	Applies a 2D or 3D transformation to an element
transform-origin	Allows you to change the position on transformed elements
transform-style	Specifies how nested elements are rendered in 3D space
perspective	Specifies the perspective on how 3D elements are viewed
perspective-origin	Specifies the bottom position of 3D elements
backface-visibility	Defines whether or not an element should be visible when not facing the screen

3D Transform Methods

Function	Description
matrix3d (*n*,*n*,*n*,*n*,*n*,*n*,*n*,*n*,*n*,*n*,*n*,*n*,*n*,*n*,*n*,*n*)	Defines a 3D transformation, using a 4x4 matrix of 16 values
translate3d(*x*,*y*,*z*)	Defines a 3D translation
translateX(*x*)	Defines a 3D translation, using only the value for the X-axis
translateY(*y*)	Defines a 3D translation, using only the value for the Y-axis

translateZ(*z*)	Defines a 3D translation, using only the value for the Z-axis
scale3d(*x*,*y*,*z*)	Defines a 3D scale transformation
scaleX(*x*)	Defines a 3D scale transformation by giving a value for the X-axis
scaleY(*y*)	Defines a 3D scale transformation by giving a value for the Y-axis
scaleZ(*z*)	Defines a 3D scale transformation by giving a value for the Z-axis
rotate3d(*x*,*y*,*z*,*angle*)	Defines a 3D rotation
rotateX(*angle*)	Defines a 3D rotation along the X-axis
rotateY(*angle*)	Defines a 3D rotation along the Y-axis
rotateZ(*angle*)	Defines a 3D rotation along the Z-axis
perspective(*n*)	Defines a perspective view for a 3D transformed element

1.23: CSS Transitions

CSS transitions allows you to change property values smoothly, over a given duration.

Transition properties:

- transition
- transition-delay
- transition-duration
- transition-property
- transition-timing-function

The following table lists all the CSS transition properties:

Property	Description	Syntax
transition	A shorthand property for setting the four transition properties into a single property	`transition: property duration timing-function delay|initial|inherit;`
transition-delay	Specifies a delay (in seconds) for the transition effect	`transition-delay: time|initial|inherit;`

transition-duration	Specifies how many seconds or milliseconds a transition effect takes to complete	`transition-duration: time|initial|inherit;`
transition-property	Specifies the name of the CSS property the transition effect is for	`transition-property: none|all|property|initial|inherit;`
transition-timing-function	Specifies the speed curve of the transition effect	`transition-timing-function: linear|ease|ease-in|ease-out|ease-in-out|step-start|step-end|steps(int,start|end)|cubic-bezier(n,n,n,n)|initial|inherit;`

How to Use CSS Transitions?

To create a transition effect, you must specify two things:

1. the CSS property you want to add an effect to
2. the duration of the effect

Note: If the duration part is not specified, the transition will have no effect, because the default value is 0.

The following example shows a 100px * 100px red element. The element has also specified a transition effect for the width property, with duration of 2 seconds:

Example

```
div {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}
```

The transition effect will start when the specified CSS property (width) changes value.
Now, let us specify a new value for the width property when a user mouses over the element:

```
<!DOCTYPE html>
<html>
<head>
<style>
        div {
          width: 100px;
```

```
    height: 100px;
    background: red;
    transition: width 2s;
   }

   div:hover {
    width: 300px;
   }
</style>
</head>
<body>

<h1>The transition Property</h1>

<p>Hover over the div element below, to see the transition effect:</p>
<div></div>

<p><b>Note:</b> This example does not work in Internet Explorer 9 and earlier versions.</p>

</body>
</html>
```


Notice that when the cursor mouses out of the element, it will gradually change back to its original style.

Specify the Speed Curve of the Transition

✓ The transition-timing-function property specifies the speed curve of the transition effect.

✓ The transition-timing-function property can have the following values:

- **ease** - specifies a transition effect with a slow start, then fast, then end slowly (this is default)
- **linear** - specifies a transition effect with the same speed from start to end
- **ease-in** - specifies a transition effect with a slow start
- **ease-out** - specifies a transition effect with a slow end
- **ease-in-out** - specifies a transition effect with a slow start and end
- **cubic-bezier(n,n,n,n)** - lets you define your own values in a cubic-bezier function

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
```

```
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}

#div1 {transition-timing-function: linear;}
#div2 {transition-timing-function: ease;}
#div3 {transition-timing-function: ease-in;}
#div4 {transition-timing-function: ease-out;}
#div5 {transition-timing-function: ease-in-out;}

div:hover {
  width: 300px;
}
</style>
</head>
<body>

<h1>The transition-timing-function Property</h1>

<p>Hover over the div elements below, to see the different speed curves:</p>

<div id="div1">linear</div><br>
<div id="div2">ease</div><br>
<div id="div3">ease-in</div><br>
<div id="div4">ease-out</div><br>
<div id="div5">ease-in-out</div><br>

<p><b>Note:</b> This example does not work in Internet Explorer 9 and earlier
versions.</p>

</body>
</html>
```

Delay the Transition Effect

- ✓ The transition-delay property specifies a delay (in seconds) for the transition effect.
- ✓ The following example has a 1 second delay before starting:

Example

```
div {
  transition-delay: 1s;
}
```

Transition + Transformation

The following example adds a transition effect to the transformation:

Example

```
div {
  transition: width 2s, height 2s, transform 2s;
}
```

More Transition Examples

The CSS transition properties can be specified one by one, like this:

Example

```
div {
  transition-property: width;
  transition-duration: 2s;
  transition-timing-function: linear;
  transition-delay: 1s;
}
```


or by using the shorthand property transition:

Example

```
div {
  transition: width 2s linear 1s;
}
```

1.24: CSS Animations

CSS animations make it possible to animate transitions from one CSS style configuration to another.

Animations consist of two components,

1. a style describing the CSS animation and

2. a set of keyframes that indicate the start and end states of the animation's style, as well as possible intermediate waypoints.

There are three key advantages to CSS animations over traditional script-driven animation techniques:

- ✓ They're easy to use for simple animations; you can create them without even having to know JavaScript.
- ✓ The animations run well, even under moderate system load. Simple animations can often perform poorly in JavaScript. The rendering engine can use frame-skipping and other techniques to keep the performance as smooth as possible.
- ✓ Letting the browser control the animation sequence lets the browser optimize performance and efficiency by, for example, reducing the update frequency of animations running in tabs that aren't currently visible.

Configuring the animation

- To create a CSS animation sequence, you style the element you want to animate with the animation property or its sub-properties.
- This lets you configure the timing, duration, and other details of how the animation sequence should progress.
- This does not configure the actual appearance of the animation, which is done using the @keyframes at-rule as described in Defining the animation sequence using keyframes below.

The animation property is a shorthand property for:

1. animation-name
2. animation-duration
3. animation-timing-function
4. animation-delay
5. animation-iteration-count
6. animation-direction
7. animation-fill-mode
8. animation-play-state

The sub-properties of the animation property are:

Value	Description	Syntax
animation-name	Specifies the name of the keyframe you want to bind to the selector	animation-name: keyframename|none|initial|inherit;

animation-duration	Specifies how many seconds or milliseconds an animation takes to complete	animation-duration: time|initial|inherit;
animation-timing-function	Specifies the speed curve of the animation	animation-timing-function: linear|ease|ease-in|ease-out|ease-in-out|step-start|step-end|steps(int,start|end)|cubic-bezier(n,n,n,n)|initial|inherit;
animation-delay	Specifies a delay before the animation will start	animation-delay: time|initial|inherit;
animation-iteration-count	Specifies how many times an animation should be played	animation-iteration-count: number|infinite|initial|inherit;
animation-direction	Specifies whether or not the animation should play in reverse on alternate cycles	animation-direction: normal|reverse|alternate|alternate-reverse|initial|inherit;
animation-fill-mode	Specifies what values are applied by the animation outside the time it is executing	animation-fill-mode: none|forwards|backwards|both|initial|inherit;
animation-play-state	Specifies whether the animation is running or paused	animation-play-state: paused|running|initial|inherit;
initial	Sets this property to its default value.	property: initial;
inherit	Inherits this property from its parent element.	property: inherit;

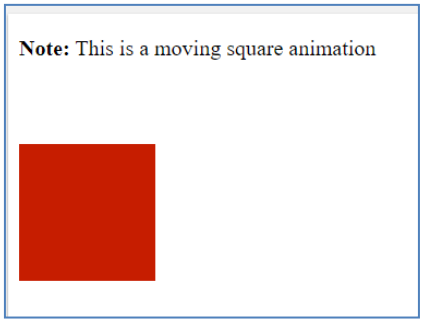
Defining the animation sequence using keyframes

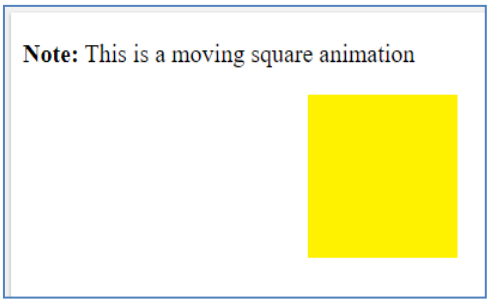
- ✓ Once you've configured the animation's timing, you need to define the appearance of the animation. This is done by establishing two or more keyframes using the **@keyframes at-rule.**
  - Each keyframe describes how the animated element should render at a given time during the animation sequence.
- ✓ Since the timing of the animation is defined in the CSS style that configures the animation, keyframes use a <percentage> to indicate the time during the animation sequence at which they take place.
  - 0% indicates the first moment of the animation sequence, while 100% indicates the final state of the animation.
- ✓ Because these two times are so important, they have special aliases: from and to. Both are optional.
  - If from/0% or to/100% is not specified, the browser starts or finishes the animation using the computed values of all attributes.

- ✓ You can optionally include additional keyframes that describe intermediate steps between the start and end of the animation.

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
        div {
          width: 100px;
          height: 100px;
          background-color: red;
          position: relative;
          animation-name: example;
          animation-duration: 4s;
          animation-iteration-count:infinite;
        }

        @keyframes example {
          0%  {background-color:red; left:0px; top:0px;}
          25% {background-color:yellow; left:200px; top:0px;}
          50% {background-color:blue; left:200px; top:200px;}
          75% {background-color:green; left:0px; top:200px;}
          100% {background-color:red; left:0px; top:0px;}
        }
</style>
</head>
<body>

<p><b>Note:</b> This is a moving square animation</p>

<div></div>

</body>
</html>
```



****

UNIT-II CLIENT SIDE PROGRAMMING

Java Script: An introduction to JavaScript–JavaScript DOM Model - Exception-Handling-Validation-Built-in-objects-Event Handling - DHTML with JavaScript- JSON introduction– Syntax – Function Files

2.1: An Introduction to JavaScript

JavaScript is the programming language of HTML and the Web, Also called as Client-Side Scripting Language designed to add interactivity to HTML pages.

- It is a Client-side script to interact with the user and make dynamic pages.
- JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java.
- But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.
- ECMA is European Computer Manufacturer's Association.

Features of JavaScript

- JavaScript is an open source scripting language.
- It is lightweight.
- It creates network-centric applications.
- It is platform independent.
- It validates form data.
- It reduces server load.

What can a JavaScript do?

- JavaScript can put dynamic text into an HTML page
- **JavaScript can react to events** - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element

- **JavaScript can read and write HTML elements** - A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data** - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

Advantages of JavaScript

- JavaScript saves server traffic.
- It performs the operations very fast.
- It is simple to learn and implement.
- It is versatile.
- JavaScript pages are executed on the client side.
- JavaScript extends its functionality to the web pages.

Disadvantages of JavaScript

- JavaScript cannot be used for networking applications.
- It doesn't have any multithreading or multiprocessor capabilities.
- It has security issues being a client-side scripting language.

What is a script?

- Script is a small, embedded program.
- The most popular scripting languages on the web are, JavaScript or VBScript.
- HTML does not have scripting capability, you need to use <script> tag.
- The <script> tag is used to generate a script.
- The </script> tag indicates the end of the script or program.

Example : Type attribute

```
<script type = “text/javascript”>
    document.write(“TutorialRide”);
</script>
```

The 'type' attribute indicates which script language you are using with the type attribute.

Example : Language attribute

```
<script language= “javascript”>
    document.write(“TutorialRide”);
</script>
```

✓ You can also specify the <script language> using the 'language' attribute.

Types of Scripts

1. Client-Side Scripting
2. Server-Side Scripting

Client-Side Scripting

- Client-Side Scripting is an important part of the Dynamic HTML (DHTML).
- JavaScript is the main client-side scripting language for the web.
- The scripts are interpreted by the browser.
- Client-Side scripting is used to make changes in the web page after they arrive at the browser.
- It is useful for making the pages a bit more interesting and user-friendly.
- It provides useful gadgets such as calculators, clocks etc.
- It enables interaction within a web page.
- It is affected by the processing speed of the user's computer.

Operation of Client-Side Scripting

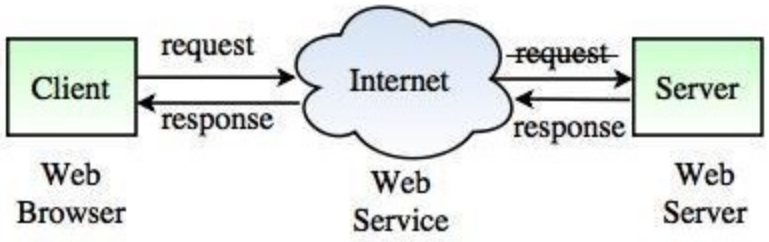


Fig. Client-Side Scripting

In the above diagram,

- The user requests a web page from the server.
- The server finds the page and sends it to the user.

- The page is displayed on the browser with any scripts running during or after the display.
- Client-Side scripting is used to make web page changes after they arrive at the browser.
- These scripts rely on the user's computer. If the computer is slow, then they may run slow.
- These scripts may not run at all if the browser does not understand the scripting language.

Server-Side Scripting

- Server-Side Scripting is used in web development.
- The server-side environment runs a scripting language which is called a web server.
- Server-Side Scripting is used to provide interactive web sites.
- It is different from Client-Side Scripting where the scripts are run by viewing the web browser, usually in JavaScript.
- It is used for allowing the users to have individual accounts and providing data from databases.
- It allows a level of privacy, personalization and provision of information that is very useful.
- It includes ASP.NET and PHP.
- It does not rely on the user having specific browser or plug-in.
- It is affected by the processing speed of the host server.

Operation of Server-Side Scripting

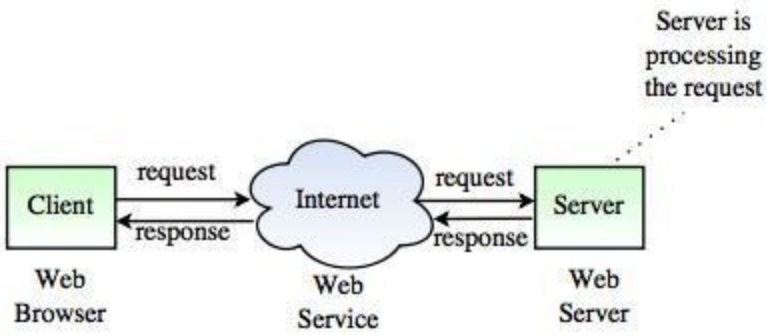


Fig. Server-Side Scripting

In the above diagram,

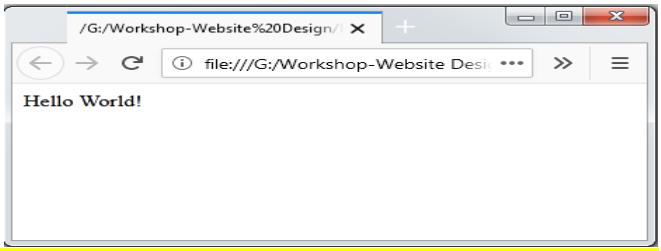
- The client requests a web page from the server.
- The script in the page is interpreted by the server, creating or changing the page content to suit the user (client) and the passing data around.
- The page in its final form is sent to the user(client) and then cannot be changed using Server-Side Scripting.
- Server-Side Scripting tends to be used for allowing the users to have individual accounts and provides data from the databases.
- These scripts are never seen by the user.
- Server-Side script runs on the server and generate results which are sent to the user.
- Running all the scripts puts a lot of load onto a server but not on the user's system.

: Syntax to write JavaScript

- In HTML, JavaScript code is inserted between <script> and </script> tags.
- <script> tag alerts the browser program to start interpreting all the text between these tags as a script (program statements).
- Old JavaScript examples may use a type attribute: <script type="text/javascript">
- The type attribute is not required. JavaScript is the default scripting language in HTML.
- Example: Displaying the string –Hello World!‖ on the browser window:

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

document.write writes a string into our HTML



Ways to write/execute JavaScript: (Where to write? JavaScript in <head> or <body>??)

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

There are three ways of executing JavaScript on a web browser.

1) Inside <HEAD> tag
2) Within <BODY> tag
3) In an External File

```
<html>
    <head>
        <title> JavaScript Demo </title>
        <script type="text/javascript">
            document.write("Hello World!");
        </script>
    </head>
    <body>
    </body>
</html>
```

JavaScript code

JavaScript code in head section

```
<html>
    <head>
        <title> JavaScript Demo </title>
    </head>
    <body>
        <script type="text/javascript">
            document.write("Hello World!");
        </script>
    </body>
</html>
```

JavaScript code

JavaScript code in body section

- ✓ Scripts can also be placed in external files:
  - External scripts are practical when the same code is used in many different web pages.
  - JavaScript files have the file extension **.js**.
  - To use an external script, put the name of the script file in the src (source) attribute of a <script> tag.
  - Example: **<script src="myScript.js"></script>**

External scripts cannot contain <script> tags.

Rules for writing the JavaScript code:

- Script should be placed inside the <script> tag.
- A semicolon at the end of each statement is optional.
- The single line comment is just two slashes (//) and multiple line comment starts with /* and ends with */.
- Use '**document.write**' for writing a string into HTML document.
- JavaScript is case sensitive.
- You can insert special characters with backslash (\& or \$).

2.1.2: Displaying Output in JavaScript

✓ **JavaScript Display Possibilities** : JavaScript can "display" data in different ways:

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

```
<html>
<body>
<h2>My First Web Page</h2>
<p>My First Paragraph.</p>
<p id="demo"></p>
<script>
        document.getElementById("demo").innerHTML = 5 + 6;
        document.write(10 + 10);
```

```
        window.alert(10 + 30);
        console.log(10 + 40);
</script>
</body>
</html>
```

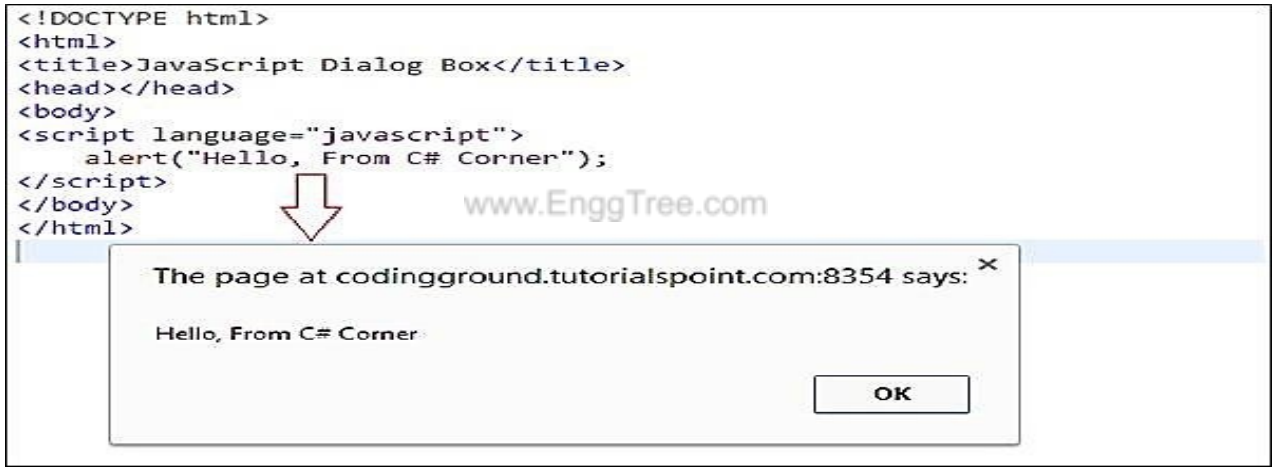
2.1.2 : JavaScript Popup/Dialog Boxes

JavaScript has three kinds of popup/dialog boxes:

1) Alert box, 2) Confirm box, and 3) Prompt box.

Alert Box

- ✓ An alert box is often used if you want to make sure information comes through to the user.
- ✓ When an alert box pops up, the user will have to click "OK" to proceed.
- ✓ Syntax; **window.alert("sometext");**



Confirm Box

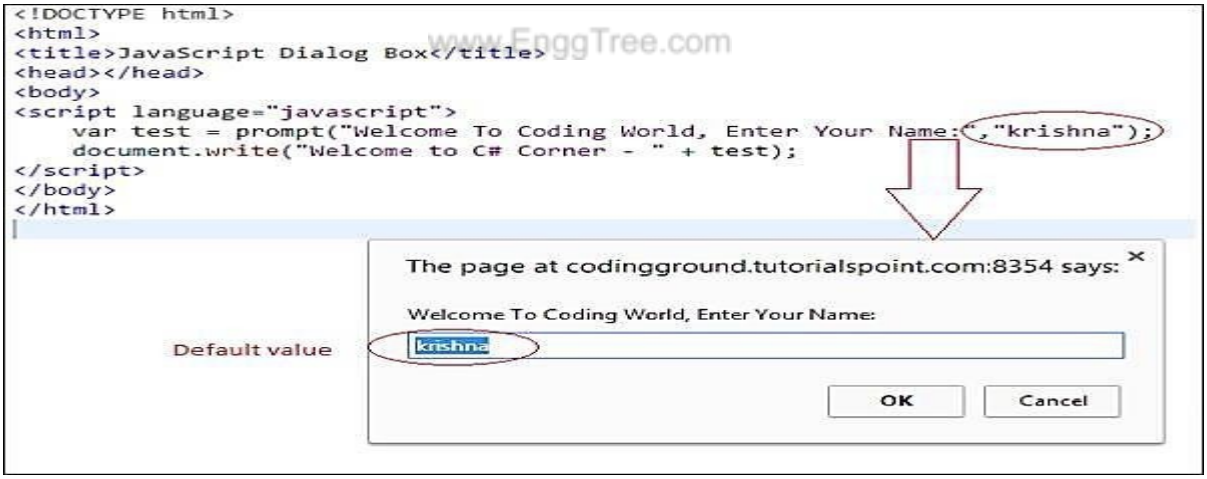
- ✓ A confirm box is often used if you want the user to verify or accept something.
- ✓ When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- ✓ If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.
- ✓ Syntax: **window.confirm("sometext");**

```
<!DOCTYPE html>
<html>
<title>JavaScript Dialog Box</title>
<head></head>
<body>
<script language="javascript">
    confirm("Do you really want to Exit?");
</script>
</body>
</html>
```

Prompt Box

- ✓ A prompt box is often used if you want the user to input a value before entering a page.
- ✓ When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- ✓ If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.
- ✓ Syntax: **window.prompt("sometext","defaultText");**

```
<!DOCTYPE html>
<html>
<title>JavaScript Dialog Box</title>
<head></head>
<body>
<script language="javascript">
    var test = prompt("Welcome To Coding World, Enter Your Name:","krishna");
    document.write("Welcome to C# Corner - " + test);
</script>
</body>
</html>
```

2.1.4: JavaScript Variables

- ✓ Variables are declared with the **var** keyword
- ✓ JavaScript variable can hold a value of any data type or expressions.
- ✓ A variable can have a short name, like x, or a more descriptive name, like carname.

Rules for JavaScript variable names:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter or the underscore character.
- Because JavaScript is case-sensitive, variable names are case- sensitive.

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables. You can declare JavaScript variables with the var keyword:

```
var x;
var carname;
```

After the declaration shown above, the variables are empty (they have no values yet). However, you can also assign values to the variables when you declare them:

```
var x=5;
var carname="Volvo";
```

After the execution of the statements above, the variable x will hold the value 5, and carname will hold the value Volvo.

Single declaration	Multiple declaration	Variable initialization
<script type = "text/javascript"> <!-- var money; var name; //--> </script>	<script type = "text/javascript"> <!– var money, name; // --> </script>	<script type = "text/javascript"> <!-- var name = "Ali"; var money; money = 2000.50; //--> </script>

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that have not yet been declared, the variables will automatically be declared. These statements:

x=5;
carname="Volvo"; have the same effect as:
var x=5;
var carname="Volvo";

Redeclaring JavaScript Variables

If you redeclare a JavaScript variable, it will not lose its original value. var x=5;
var x;

After the execution of the statements above, the variable x will still have the value of 5. The value of x is not reset (or cleared) when you redeclare it.

The Lifetime of JavaScript Variables

- If you declare a variable within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called **local variables.** You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.

- If you declare a variable outside a function, all the functions on your page can access it. These variables are called **global variables.** The lifetime of these variables starts when they are declared, and ends when the page is closed.

2.1.3: JavaScript Operators

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division

%	Modulus (Reminder)
++	Increment
--	Decrement

Comparison Operators

Operator	Description
= =	Equal To
= = =	Exactly Equal To
!=	Not Equal To
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To

Assignment Operators

Operator	Description
=	Simple Assignment
+=	Add and Assignment
-=	Subtract and Assignment
*=	Multiply and Assignment
/=	Divide and Assignment
%=	Modulus and Assignment

Logical Operators

Operator	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left Shift
>>	Right Shift
>>>	Right Shift with Zero

Special Operators

Operator	Description
NEW	Creates an instance of an object type.
DELETE	Deletes property of an object.
DOT(.)	Specifies the property or method.
VOID	Does not return any value. Used to return a URL with no value.

2.1.3: JavaScript Control and Looping Structure

1) If Statement

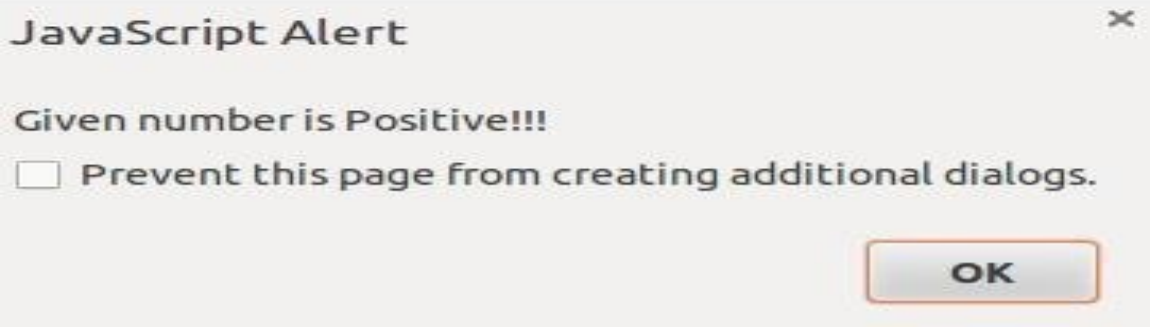
- IF statement is a conditional branching statement.
- In 'IF' statement, if the condition is true a group of statement is executed. And if the condition is false, the following statement is skipped.

```
if(condition)
{
    //Statement 1;
    //Statement 2;
}
```

Example : Simple Program for IF Statement

```
<html>
    <body>
    <script type="text/javascript">
        var num = prompt("Enter
        Number"); if (num > 0)
        {
            alert("Given number is Positive!!!");
        }
    </script>
    </body>
</html>
```

Output:



2) If – Else Statement

- If – Else is a two-way decision statement.
- It is used to make decisions and execute statements conditionally.

Flow Diagram of If – Else Statement

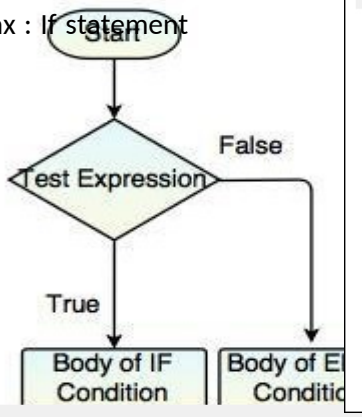


Fig. Flow Diagram of IF - ELSE Statement

Syntax : If statement

Syntax : If-Else statement if (condition)

```
{
//Statement 1;
}
else if(condition)
{
//Statement 2;
}
else
{
//Statement 3;
}
```

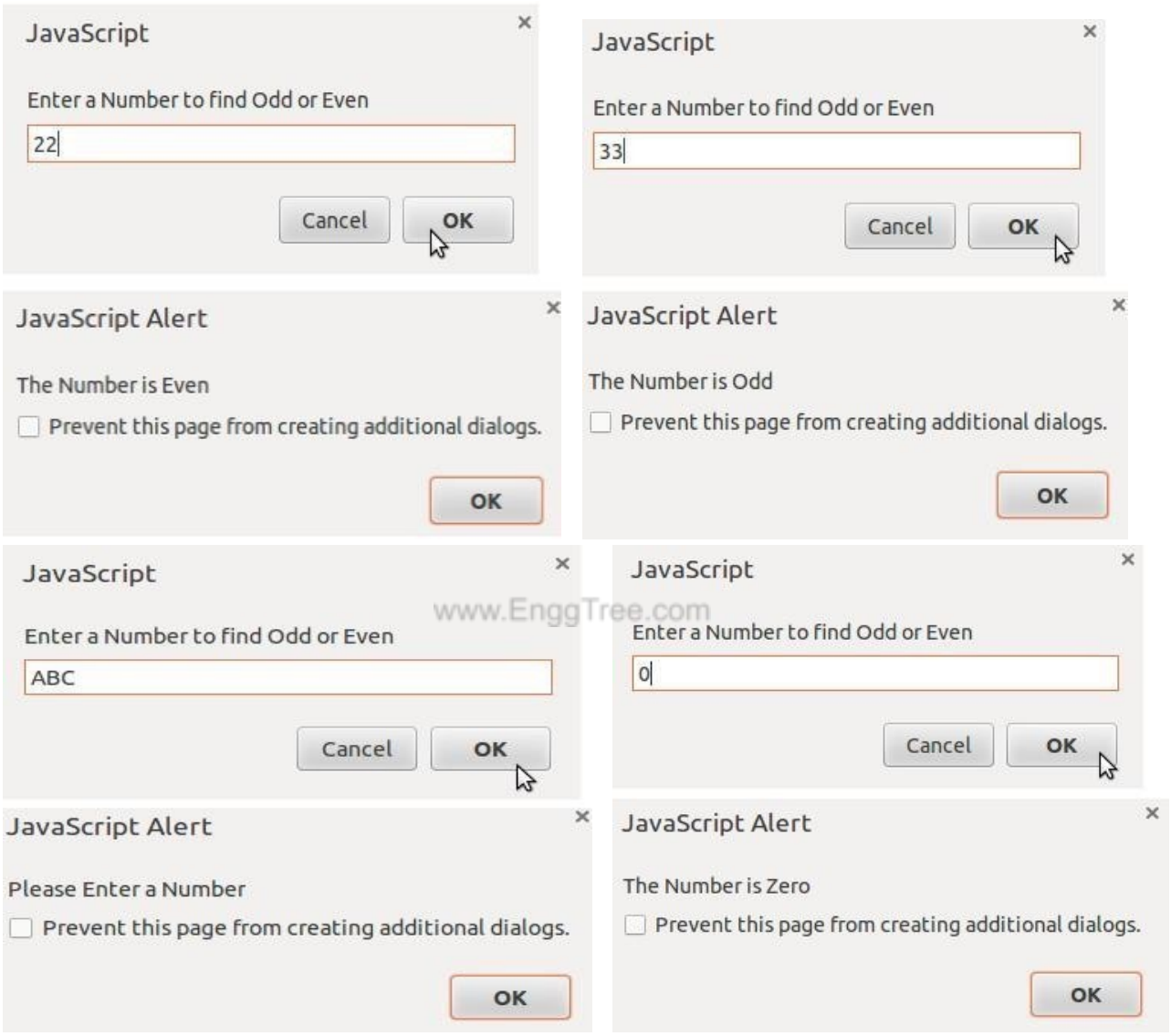


```
<html>
    <head>
    <script type="text/javascript">
        var no = prompt("Enter a Number to find Odd or Even");
        no = parseInt(no);
        if (isNaN(no))
        {
            alert("Please Enter a Number");
        }
        else if (no == 0)
        {
            alert("The Number is Zero");
        }
        else if (no % 2)
        {
            alert("The Number is Odd");
        }
```

```
else
{
    alert("The Number is Even");
}
```

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

Output



3) Switch Statement

- Switch is used to perform different actions on different conditions.
- It is used to compare the same expression to several different values.

Flow Diagram of Switch Statement

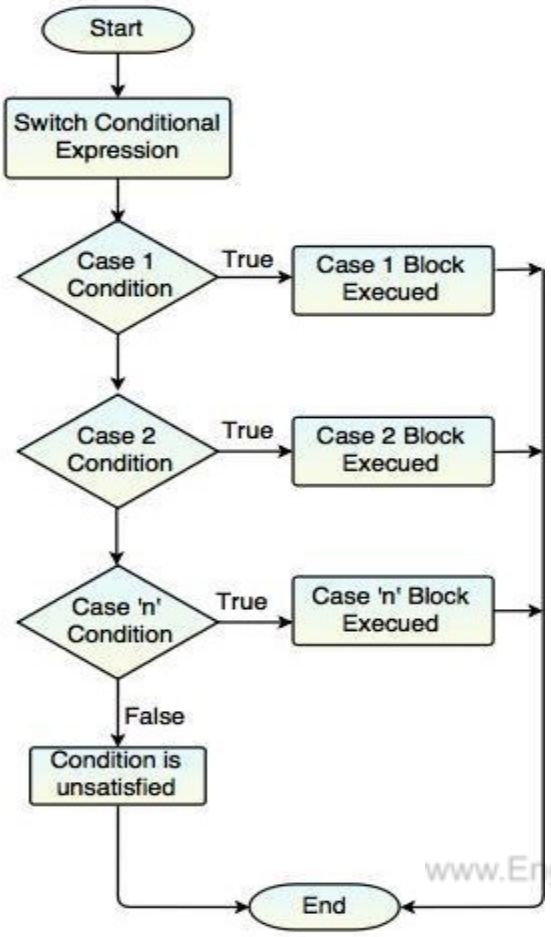


Fig. Flow Diagram of Switch Statement

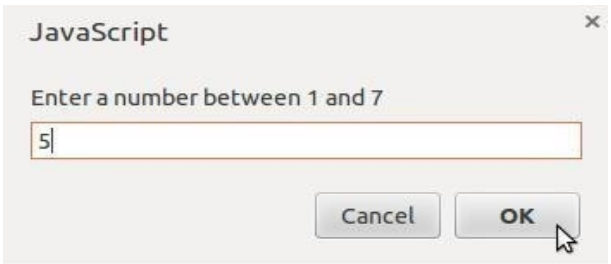
Example : Simple Program for If-Else Statement

Example : Simple Program for Switch Statement

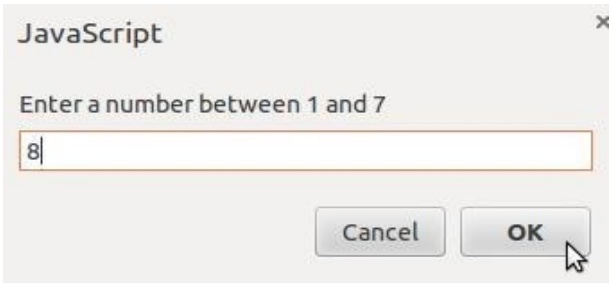
```
<html>
<head>
    <script type="text/javascript">
    var day = prompt("Enter a number between 1 and 7");
    switch (day)
    {
        case (day="1"):
            document.write("Sunday");
            break;
        case (day="2"):
            document.write("Monday");
            break;
```

```
        case (day="3"):
            document.write("Tuesday")
            ; break;
        case (day="4"):
            document.write("Wednesday");
            break;
        case (day="5"):
            document.write("Thursday");
            break;
        case (day="6"):
            document.write("Friday");
            break;
        case (day="7"):
            document.write("Saturday");
            break;
        default:
            document.write("Invalid
            Weekday"); break;
    }
    </script>
</head>
</html>
```


Output:



Thursday

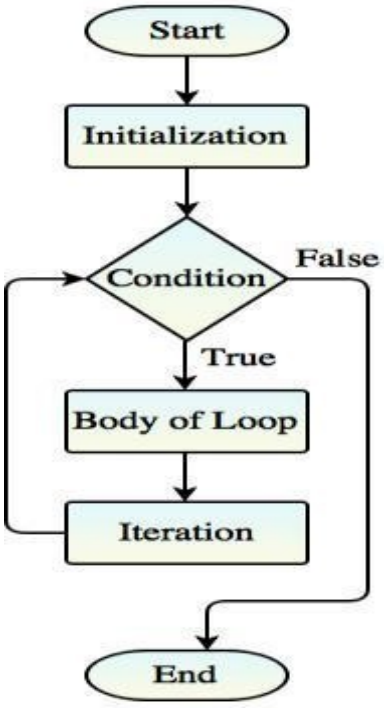


Invalid Weekday

4) For Loop

- For loop is a compact form of looping.
  It includes three important parts:
  1. Loop Initialization
  2. Test Condition
  3. Iteration
- All these three parts come in a single line separated by semicolons(;).

Flow Diagram of 'For' Loop



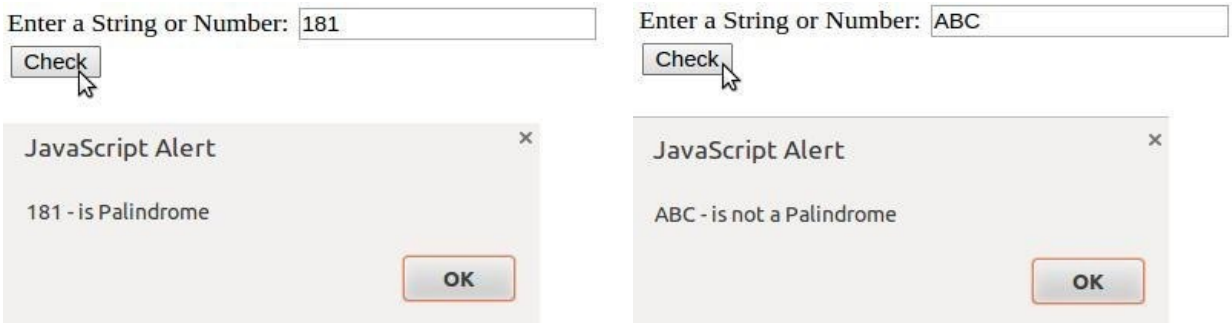

```
Syntax switch(expression)
{
case condition 1:
//Statements; break;
case condition 2:
//Statements; break;
case condition 3:
//Statements; break;
.
.
case condition n:
//Statements; break;
default:
//Statement;
}
```


Example : Palindrome Program using For Loop

```
<html>
<body>
    <script
    type="text/javascript">
    function palindrome()
    {
        var revstr = " ";
        var strr = document.getElementById("strr").value;
        var i = strr.length;
        for(var j=i; j>=0; j--)
        {
```

```
            revstr = revstr+strr.charAt(j);
        }
        if(strr == revstr)
        {
            alert(strr+" - is Palindrome");
        }
        else
        {
            alert(strr+" - is not a Palindrome");
        }
    }
    </script>
    <form>
        Enter a String or Number: <input type="text" id="strr"
name="checkpalindrome"><br>
        <input type="submit" value="Check" onclick="palindrome();">
    </form>
</body>
</html>
```

Output:



5) For-in Loop

- For-in loop is used to traverse all the properties of an object.
- It is designed for looping through arrays.

Syntax for (var

```
{
    //Statements;
}
```

6) While Loop

- While loop is an entry-controlled loop statement.
- It is the most basic loop in JavaScript.
- It executes a statement repeatedly as long as expression is true.
- Once the expression becomes false, the loop terminates.

Flow Diagram of While Loop

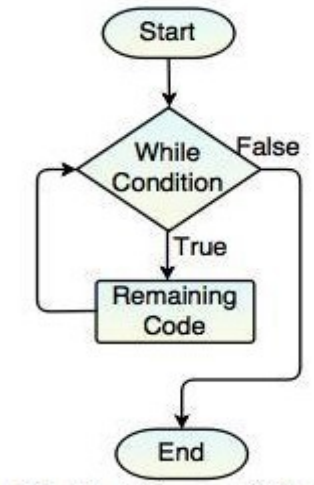


Fig. Flow Diagram of While Loop

Syntax

```
for(initialization; test-condition; increment/decrement)
{
Syntax
//Statements;
}
```


Example : Fibonacci Series Program using While Loop

```
<html>
<body>
    <script
        type="text/javascript"> var
        no1=0,no2=1,no3=0;
        document.write("Fibonacci Series:"+"<br>");
        while (no2<=10)
        {
            no3 =
            no1+no2; no1 =
            no2;
            no2 = no3;
            document.write(no3+"<br>");
```

Fibonacci Series:
1
2
3
5
8
13

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

7) Do-While Loop

- Do-While loop is an exit-controlled loop statement.
- Similar to the While loop, the only difference is condition will be checked at the end of the loop.
- The loop is executed at least once, even if the condition is false.

Flow Diagram of Do – While

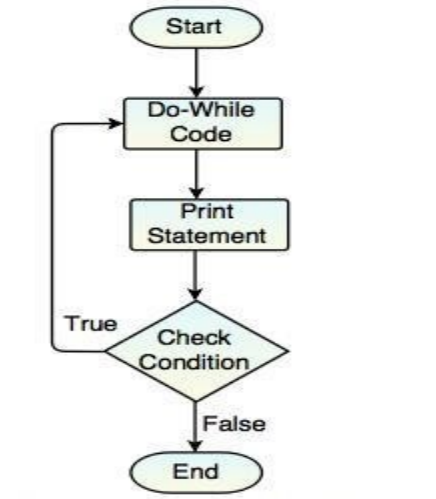


Fig. Flow Diagram of Do-While Loop

```
while (condition)
{
//Statements;
}
```


Example : Simple Program on Do-While Loop

```
<html>
<body>
    <script type ="text/javascript">
    var i = 0;
    do
    {
        document.write(i+"<br>") i+
        +;
    }
    while (i <= 5)
    </script>
```

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

Output:

0
1
2
3
4
5

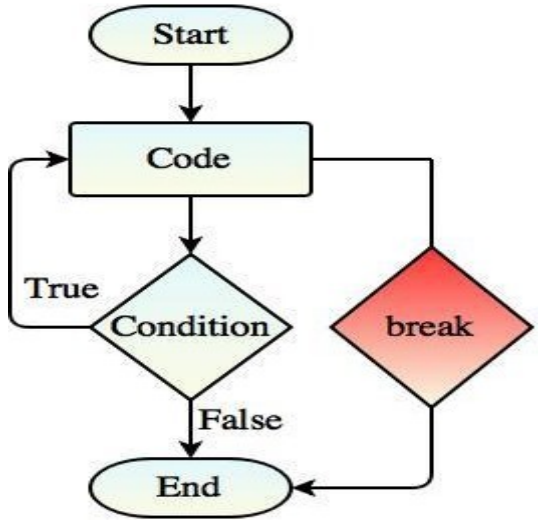
Difference between While Loop and Do – While Loop

While Loop	Do – While Loop
In while loop, first it checks the condition and then executes the program.	In Do – While loop, first it executes the program and then checks the condition.
It is an entry – controlled loop.	It is an exit – controlled loop.
The condition will come before the body.	The condition will come after the body.
If the condition is false, then it terminates the loop.	It runs at least once, even though the conditional is false.
It is a counter-controlled loop.	It is a iterative control loop.

8) Break Statement

- Break statement is used to jump out of a loop.
- It is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Diagram of Break Statement



Syntax

```
do
{
    //Statements;
```

9) Continue Statement

- Continue statement causes the loop to continue with the next iteration.
- It skips the remaining code block.

Flow Diagram of Continue Statement

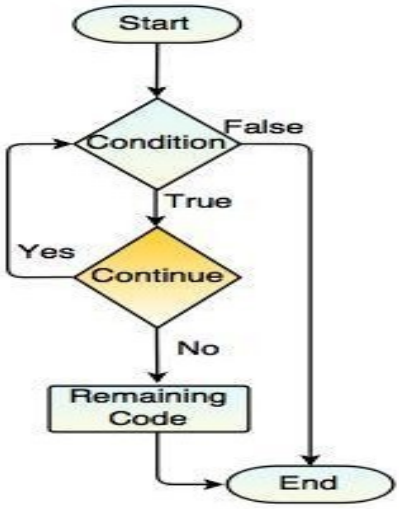


Fig. Flow Diagram of Continue Statement

Syntax:

```
break;
```

2.1.3: JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).

JavaScript Built-in Functions

- JavaScript provides number of built-in functions.

Common Built-in Functions

Functions	Description
isNan()	**Returns true,** if the object is Not a Number. **Returns false,** if the object is a number.
parseFloat (string)	If the string begins with a number, the function reads through the string until it finds the end of the number; cuts off the remainder of the string and returns the result. If the string does not begin with a number, the function returns NaN.
parseInt (string)	If the string begins with an integer, the function reads through the string until it finds the end of the integer, cuts off the remainder of the string and returns the result. If the string does not begin with an integer, the function returns NaN (Not a Number).
String (object)	Converts the object into a string.
eval()	Returns the result of evaluating an arithmetic expression.

User-defined Functions

- User-defined function means you can create a function for your own use. You can create yourself according to your need.
- In JavaScript, these functions are written in between the <HEAD> tag of the HTML page.

JavaScript Function Syntax

- function keyword --- followed by a name ---- followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs
- (same rules as variables).
- The parentheses may include parameter names separated by commas:
- (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets: {}
- **Syntax:**

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

Example :

```
function myFunction(p1, p2)
{
  return p1 * p2;  // The function returns the product of p1 and p2
}
```

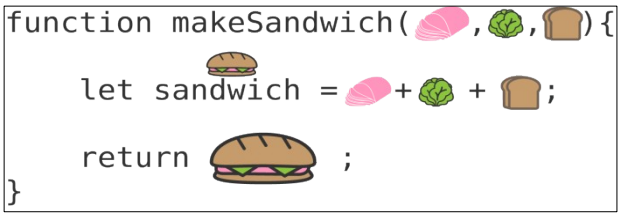

Function Invocation

- ✓ The code inside the function will execute when "something" invokes (calls) the function:
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

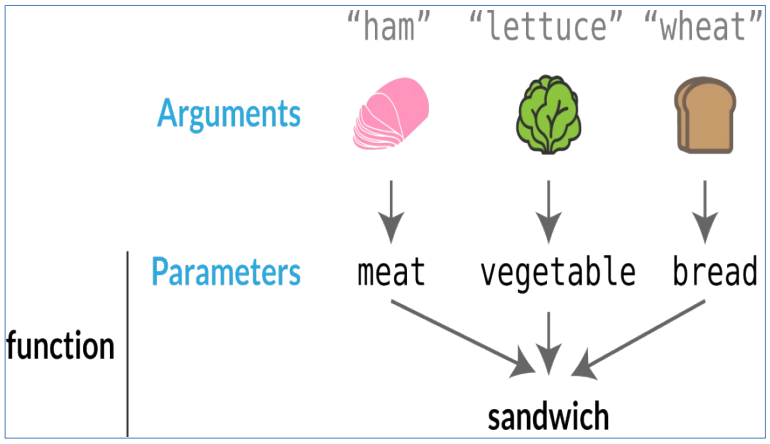
Function Return

- ✓ When JavaScript reaches a return statement, the function will stop executing.
- ✓ If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- ✓ Functions often compute a return value. The return value is "returned" back to the "caller–.

Example:

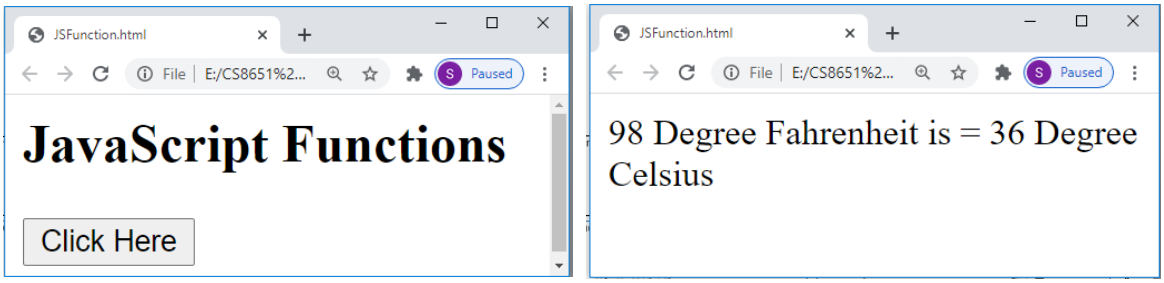



```
function makeSandwich(meat,vegetable,bread){

    let sandwich = meat + vegetable + bread;

    return sandwich;
}
```



Example:

```
<html>
<head>
<script>

function toCelsius(f) {
  var res=(5/9) * (f-32);
 document.write(f+ " Degree Fahrenheit is ="+Math.floor(res,2)+" Degree

Celsius");
}

</script>
</head>
<body>
<h2>JavaScript Functions</h2>
<button onclick='toCelsius(98);'>Click Here</button>
</body>
</html>
```



2.1.3: JavaScript Arrays

- Array is a grouping of objects.
- It stores multiple values in a single variable.
- It stores a fixed-size sequential collection of elements of the same type.
- It is used to store collection of data.

Creating and Initializing Arrays

```
var variable_name = new Array();     // Creating an
Array var arr = [];          // Creating an Array
var arr1 = [1, 2, 3];              // Initializing an Array
```


Multidimensional Array

```
var arr2 = [
                [1, 2, 3] ,
                ['a', 'b', 'c']
                ,
            ]; ['x', 'y', 'z']
```

Iteration through Arrays

```
function show_array(arr)
{
    for (var i = 0; i < arr.length; i++ )
    {
        document.write(array[i]);
        document.write('<br/>');
```

```
    }
}
var arr1 = [1, 2,
3];
show_array(arr1);
```

Array Properties

Array Properties	Description
Constructor	It returns a reference to the array function that created the object.
Index	It represents the zero-based index of the match in the string.
Length	It reflects the number of elements in an array.
Input	It presents only an array created by regular expression matches.
Prototype	It allows you to add properties and methods to an object.

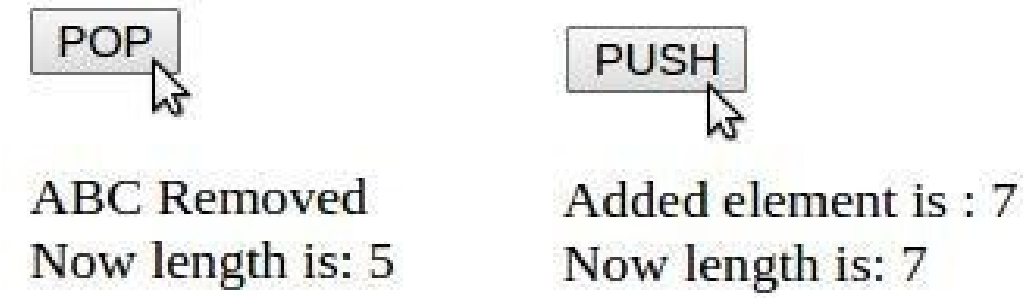
Array Methods

Methods	Description
concat()	It returns a new array comprised of this array joined with other arrays and values.
every()	It returns true if every element in this array satisfies the provided testing function.
filter()	It creates a new array with all the elements of this array for which the provided filtering function returns true.
indexOf()	It returns the first index of an element within the array equal to the specified value.
join()	It joins all elements of an array into a string.
pop()	It removes the last element from an array and returns that element.
push()	It adds one or more elements to the end of an array and returns the new length of the array.
reverse()	It reverses the order of the elements of an array.
sort()	It represents the source code of an object.

Example: Program on Array Methods – POP() & PUSH()

```
<html>
    <body>
        <button onclick="arrpop();">POP</button>
        <button onclick="arrpush();"> PUSH </button>
        <script>
            function arrpop()
            {
                var numbers = ["1", "2", "3", "4", "5", "ABC"];
                document.write(numbers.pop()+"  "+"Removed"+"<br>");
                //pop() removes the last element of an array

                // Now we have [1,2,3,4,5]
               document.write("Now length is: "+numbers.length); // ABC removed
            }
            function arrpush()
            {
                var numbers = ["1","2","3","4","5","6"]
                numbers.push("7");
                // now we got ["1","2","3","4","5","6"]
                document.write("Added element is :"+" "+numbers[numbers.length-1])
                // Now we have [1,2,3,4,5,6]
                document.write("<br>"+"Now length is: "+numbers.length); // 7 Added
            }
        </script>
    </body>
</html>
```

Output:



Syntax:

Explain Document Tree with an example. (8) (May / June 2011, May / June 2012)
How DOM can be used to access and modify the HTML documents? Explain.

Definition: DOM

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

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

Various DOM Levels:

- DOM 0 – supported by early browsers. This could support JavaScript.
- DOM 1 – released in 1998 which was focused on XHTML and HTML.
- DOM 2 – Released in 2000. Supports stylesheets, event model and traversal within the documents.
- DOM 3 – Current release published in 2004. It could deal with XML with DTD and schema, document validations.

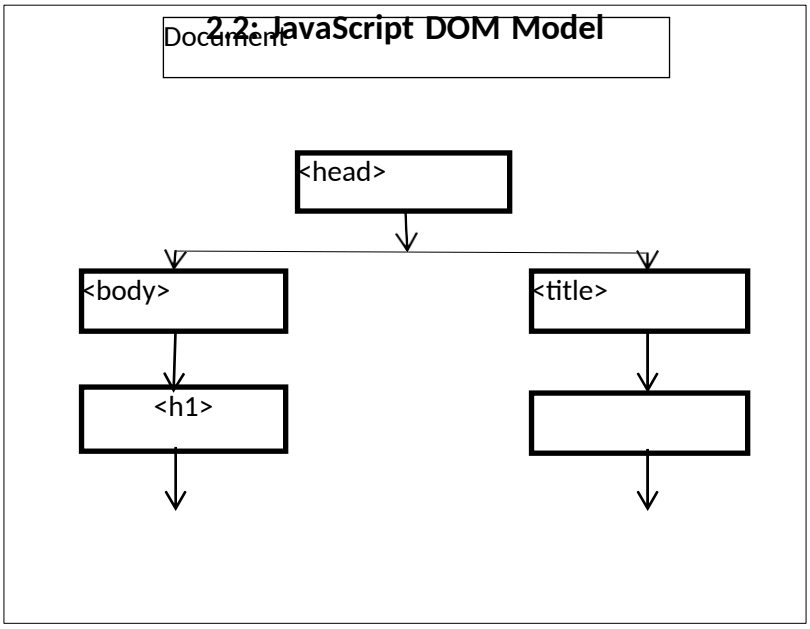
The DOM Tree:

In DOM, when the HTML or XML document's elements are syntactically correct, the documents are represented as tree structure in which every element is represented as nodes. This tree structure is called as **DOM Tree.**

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

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

```
  </head>
  <body>
    <h1> Hello! </h1>
   </body>

</html>
```



2.2: JavaScript DOM Model

- At the top level, there is an html node, with two children: head and body, among which only head has a child tag title.
- HTML tags are *element nodes* in DOM tree, pieces of text become *text nodes*. Both of them are *nodes*, just the type is different.

My PageHello!

ACCESSING DOM:

- ✓ The HTML DOM can be accessed with JavaScript (and with other programming languages).
- ✓ In the DOM, all HTML elements are defined as **objects**.
- ✓ With the object model, JavaScript gets all the power it needs to create dynamic HTML:
  - JavaScript can change all the HTML elements in the page
  - JavaScript can change all the HTML attributes in the page
  - JavaScript can change all the CSS styles in the page
  - JavaScript can remove existing HTML elements and attributes
  - JavaScript can add new HTML elements and attributes

- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

Types of nodes

There are many types of nodes in the DOM document tree that specifies what kind of node it is. Every Object in the DOM document tree has properties and methods defined by the Node host object.

The following table lists the non method properties of Node object.

TABLE 5.2: Non-method properties of Node instances.

Property	Description
nodeType	Number representing the type of node (Element, Comment, etc.).
nodeName	String providing a name for this Node (form of name depends on the nodeType; see text).
parentNode	Reference to object that is this node's parent.
childNodes	Acts like a read-only array containing this node's child nodes. Has length 0 if this node has no children.
previousSibling	Previous sibling of this node, or null if no previous sibling exists.
nextSibling	Next sibling of this node, or null if no next sibling exists.
attributes	Acts like a read-only array containing Attr instances representing this node's attributes.

The following table lists the node types commonly encountered in HTML documents and the nodeType value for each one.

Node Type	nodeType constant	nodeType value
Element	Node.ELEMENT_NODE	1
Text	Node.TEXT_NODE	3
Document	Node.DOCUMENT_NODE	9
Comment	Node.COMMENT_NODE	8
DocumentFragment	Node.DOCUMENT_FRAGMENT_NODE	11
Attr	Node.ATTRIBUTE_NODE	2

The following table lists the method properties of Node object.

TABLE 5.4: Method properties of Node instances.

Method	Functionality
hasAttributes()	Returns Boolean indicating whether or not this node has attributes.
hasChildNodes()	Returns Boolean indicating whether or not this node has children.
appendChild(Node)	Adds the argument Node to the end of the list of children of this node.
insertBefore(Node, Node)	Adds the first argument Node in the list of children of this node immediately before the second argument Node (or at end of child list if second argument is null).
removeChild(Node)	Removes the argument Node from this node's list of children.
replaceChild(Node, Node)	In the list of children of this node, replace the second argument Node with the first.

The HTML DOM Document:

In the HTML DOM object model, the document object represents our web page.

The document object is the owner of all other objects in our web page.

Finding HTML Elements:

Often, with JavaScript, we may want to manipulate HTML elements.

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

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

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

Changing HTML Elements

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

Changing the Value of an Attribute:

To change the value of an HTML attribute, use this syntax:

document.getElementById(*id*).*attribute=new value*

Adding and Deleting Elements

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

Adding Events Handlers

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

Reacting to Events

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

onclick=*JavaScript*

Examples of HTML events:

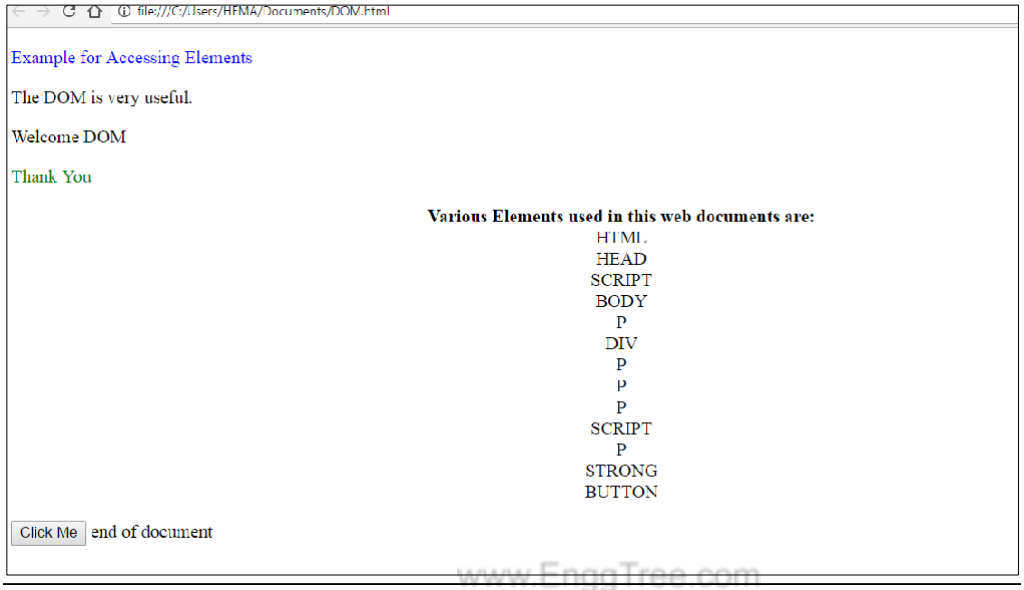
- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

Example:

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

```
<script type="text/javascript">
  var  page_element="";
  function end()
  {
      var txt=document.createTextNode("end of document");
      document.body.appendChild(txt);
  }
  function Display()    // function definition
  {
   for(i=0;i<document.all.length;i++)
{
       page_element+="<br>"+document.all[i].tagName;  // accessing all the
elements using DOM
}
      pmsg.align="center";   // accessing element's atrribute
      pmsg.innerHTML+=page_element;// setting the element's text
  }
  </script>
</head>
<body onload="Display()">
<p class="exp">Example for Accessing Elements</p>
<div id="main">
<p>The DOM is very useful.</p>
<p id="demo"></p>
</div>
<p class="exp">Thank You</p>
<script>
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
y[1].innerHTML = "Welcome DOM";
var z=document.getElementsByClassName("exp");
```

```
z[0].style.color="blue";
z[1].style.color="Green"
;
</script>
<p id="pmsg" > <strong> Various Elements used in this web documents are:</strong></p>
<button id="btn" value="click me" onClick="end()">Click Me</button>
</body></html>
```



DOM Tree for the Document

What is form validation?

- Form validation is the process of checking the forms that have been filled in correctly before they are processed.
- It provides a method to check the user entered information on client-side before the data is submitted to the server-side.

- **It includes two methods for validating forms:**
  1. Server-Side (ASP, PHP)
  2. Client-Side (JavaScript)
- It displays alerts for incorrect data entered by the user.
- Client-side validation is faster than Server-side validation.

Example : Simple Form Validation Program

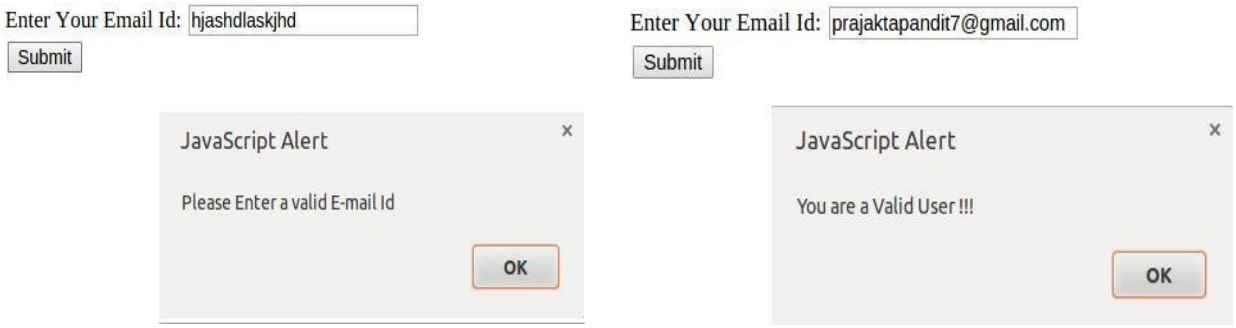
validation.html **//File name**

```
<html>
<body>
    <script>
    function validateemail()
    {
        var a =
        document.myform.email.value; var
        atposition = a.indexOf("@");
        var dotposition = a.lastIndexOf(".");
        if (atposition<1 || dotposition<atposition+2 ||
                                    dotposition+2>=a.length)
        {
            alert("Please Enter a valid E-mail
            Id"); return false;
        }
    }
    </script>
</body>
<body>
    <form name="myform" method="post" action="validpage.html"
            onsubmit="return validateemail();">
        Enter Your Email Id: <input type="text" name="email"><br/>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```


validpage.html **//File name**

```
<html>
    <body>
    <script type="text/javascript">
        alert("You are a Valid User
        !!!");
    </script>
    </body>
</html>
```

Output:



Example 2: Form Validation

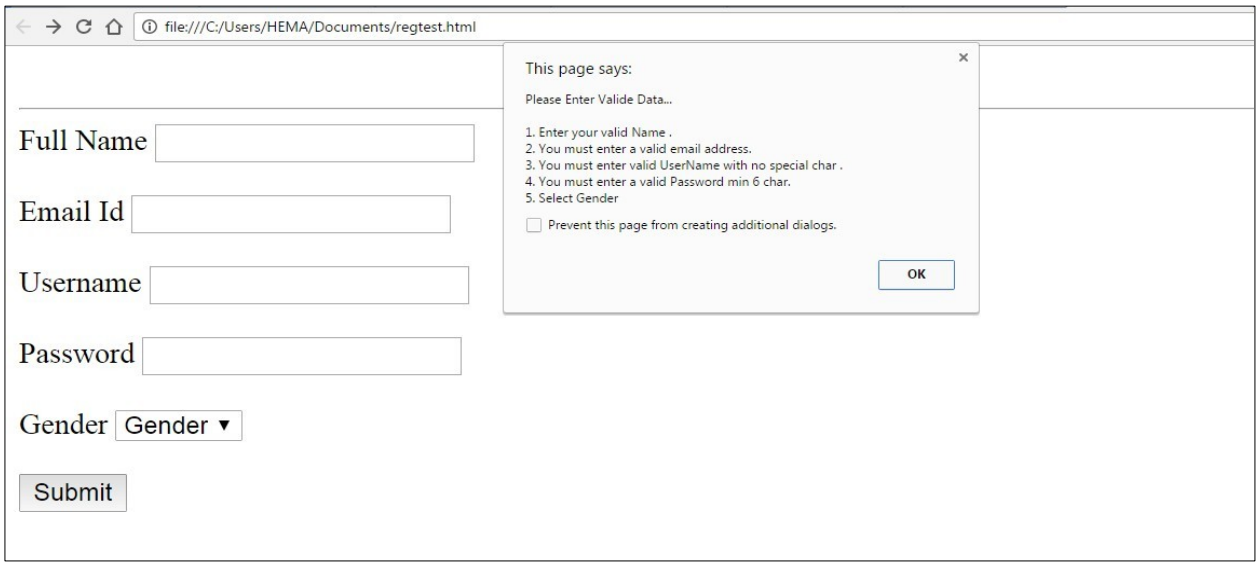
```
<!DOCTYPE html >
<html>
<head>
<title> Registration Form l</title>
<!-- Meta Tags -->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<!-- JavaScript -->
```



```
<script type="text/javascript">
var ck_name = /^[A-Za-z0-9 ]{3,20}$/;
var ck_email = /^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-
z]{2,6}(?:\.[a-z]{2})?)$/i
var ck_username = /^[A-Za-z0-9_]{1,20}$/;
var ck_password = /^[A-Za-z0-9!@#$%^&*()_]{6,20}$/;

function validate(){
var name = form.name.value;
 var email =
 form.email.value;
 var username =
 form.username.value; var password =
 form.password.value; var gender =
 form.gender.value;
 var errors = [];
```

```
 if  (!ck_name.test(name))  {
  errors[errors.length]  =  "Enter  your  valid  Name  .";       }

 if  (!ck_email.test(email))  {
  errors[errors.length] = "You must enter a valid email address.";
 }
 if   (!ck_username.test(username))   {
  errors[errors.length] = "You must enter valid UserName with no special
char .";
 }

 if  (!ck_password.test(password))  {
  errors[errors.length] = "You must enter a valid Password min 6 char.";
 }
 if (gender==0) {
  errors[errors.length] = "Select Gender";
 }

  if (errors.length > 0) {
  reportErrors(errors);
  return false;
 }
  return true;
}

function reportErrors(errors){
 var msg = "Please Enter Valide Data...\n";
 for (var i = 0; i<errors.length; i++) {
  var numError = i + 1;
  msg += "\n" + numError + ". " + errors[i];
 }
 alert(msg);
}
</script>
</head>
```

```
<body>

<center> REGISTRATION FORM</center> <hr>
<form action="thanks.html" name="form">

  <label> Full Name </label>
  <input type="text" name="name" value="" /> <br><br>

  <label> Email Id </label>
  <input type="text" name="email" value="" /> <br><br>

  <label> Username </label>
  <input type="text" name="username" value="" /> <br><br>

  <label> Password </label>
  <input type="text" name="password" value="" /> <br><br>

  <label> Gender </label>
  <select name="gender">
    <option value="0">Gender</option>
    <option value="1">Female</option>
    <option value="2">Male</option>
  </select> <br><br>
  <input type="submit" value="Submit" onclick="return validate()">
</form>
</body>
</html>
```



2.5: Form

Explain the DOM Event handling with suitable example.
(May / June 2011, Nov / Dec 2011, May / June 2012)

Events: An event is an activity that represents a change in the environment. JavaScript events allow scripts to respond to user interactions and modify the page accordingly.
Example for events: mouse clicks, pressing a key.

Event Handlers: Functions that handle events are called **event handlers.** They contain the script that gets executed in response to the events.

Advantage of Event Handling: Events and Event Handler makes web applications more **responsive, dynamic and interactive.**

List of Events (Intrinsic Event Attributes):

Intrinsic Event Attributes: Intrinsic Event Attribute is an attribute associated with a HTML element along with an event and the javascript function (Event Handler) to handle the event.

Event handler	Applies to:	Triggered when:
onAbort	Image	**The loading of the image is cancelled.**
onBlur	Button, Checkbox, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	**The object in question loses focus (e.g. by clicking outside it or pressing the TAB key).**
onChange	Select, Text, TextArea	**The data in the form element is changed by the user.**
onClick	Button, Checkbox, Link, Radio, Reset, Submit	**The object is clicked on.**
onDblClick	Document, Link	**The object is double-clicked on.**

onError	Image	**A JavaScript error occurs.**
onFocus	Button, Checkbox, Password, Radio, Reset, Select, Submit, Text, TextArea	**The object in question gains focus (e.g. by clicking on it or pressing the TAB key).**
onKeyDown	Image, Link, TextArea	**The user presses a key.**
onKeyPress	Image, Link, TextArea	**The user presses or holds down a key.**
onKeyUp	Image, Link, TextArea	**The user releases a key.**
onLoad	Image, Window	**The whole page has finished loading.**
onMouseDown	Button, Link	**The user presses a mouse button.**
onMouseMove	None	**The user moves the mouse.**
onMouseOut	Image, Link	**The user moves the mouse away from the object.**
onMouseOver	Image, Link	**The user moves the mouse over the object.**
onMouseUp	Button, Link	**The user releases a mouse button.**
onMove	Window	**The user moves the browser window or frame.**
onReset	Form	**The user clicks the form's Reset button.**
onResize	Window	**The user resizes the browser window or frame.**
onSelect	Text, Textarea	**The user selects text within the field.**
onSubmit	Form	**The user clicks the form's Submit button.**
onUnload	**Window**	**The user leaves the page.**

Registering Event Handler:

Assigning an event handler to an event on a DOM node is called **registering an event handler.**

Two ways of registration:

1. Inline model: Treating events as attributes of HTML elements. These event attributes are called as **intrinsic event attributes.**

Example: <p onclick="myfunction()">

Where,

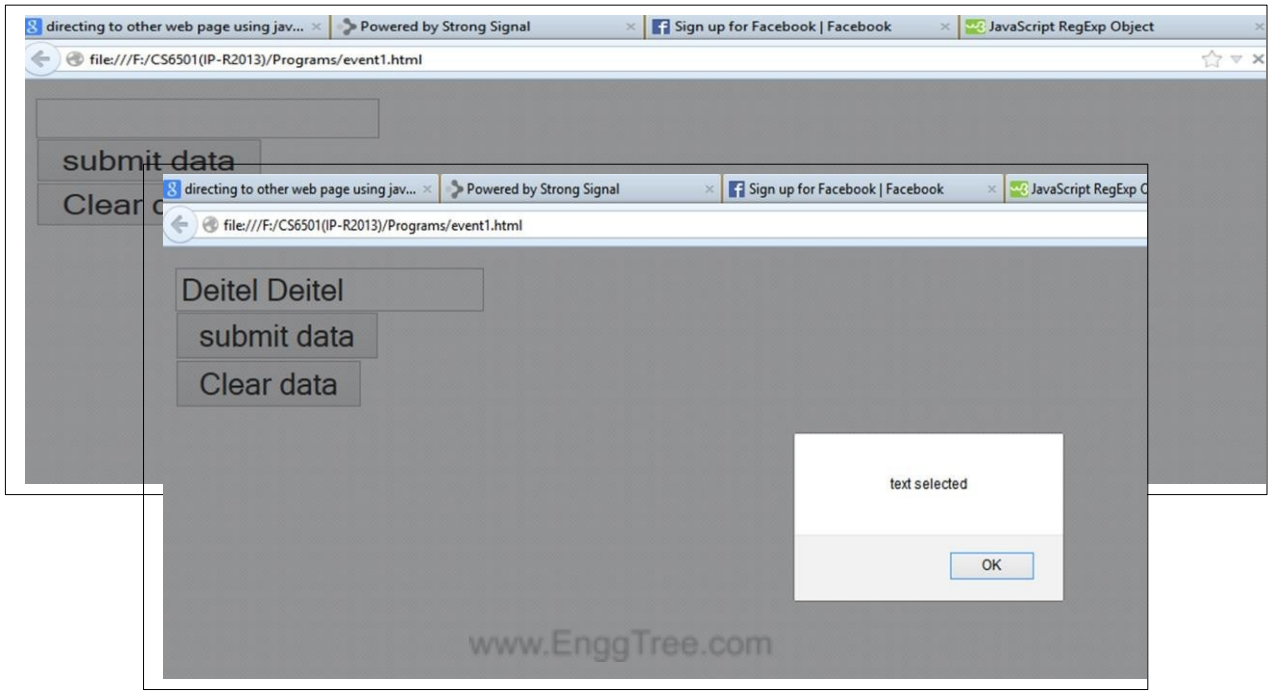
Onclick – intrinsic event attribute.

Myfunction() – event handler to handle the event.

2. Traditional Model: Registering event handler through DOM. Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Event Handling in JavaScript</title>
   <script type="text/javascript">
     function handleSubmit()
     {
       window.alert("Data Successfully submitted!");
     }
     function handleReset()
     {
       window.alert("Clearing Form Data !.....");
     }
     function registerEvent()
     {
       var reset=document.getElementById("clear");
        reset.onclick=handleReset;
     }
   </script>
   </head>
   <body onload="window.alert('Welcome! opening your
                        page'); registerEvent()">

       <form onsubmit="handleSubmit();">
         <input type="text" onselect="window.alert('text selected')" /><br />
```

```
        <input type="submit" value="submit data" /><br />
        <input type="reset" id="clear" value="Clear data" /><br />
      </form>
  </body>
</html>
```



Example 2 : Simple Program on onload() Event handler

```
<html>
    <head>
    <script
    type="text/javascript">
    function time()
    {
        var d = new Date();
        var ty = d.getHours() + ":"+d.getMinutes()+":"+d.getSeconds();
        document.frmty.timetxt.value=ty;
        setInterval("time()",1000)
    }
    </script>
```

```
    </head>
<body onload="time()">
    <center><h2>Displaying Time</h2>
        <form name="frmty">
            <input type=text name=timetxt size="8">
        </form>
    </center>
</body>
</html>
```

Output:

Example 3: Simple Program on onsubmit() & onfocus() Event handler

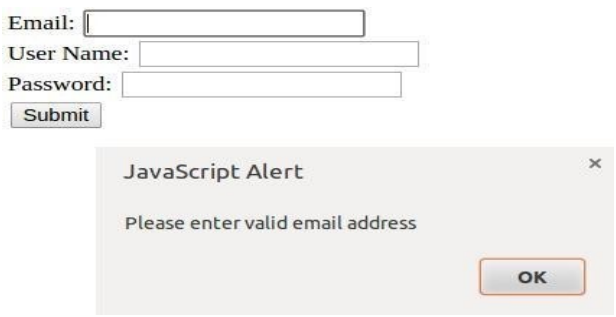
```
<html>
    <body>
        <script>
            function validateform()
            {
                var  uname=document.myform.name.value;
                var  upassword=document.myform.password.value;
                if (uname==null || uname=="")
                {
                    alert("Name cannot be left
                    blank"); return false;
                }
                else if(upassword.length<6)
                {
                    alert("Password must be at least 6 characters long.");
                    return false;
                }
            }
```

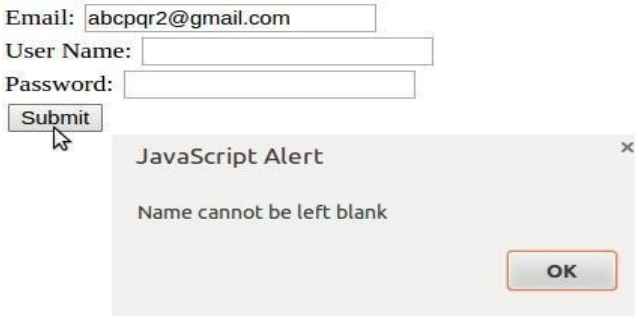
```
            function emailvalidation()
            {
                var a=document.myform.email.value
                if (a.indexOf("@")==-1)
                {
                    alert("Please enter valid email
                    address")
                    document.myform.email.focus()
                }
            }
        </script>
    <body>
        <form name="myform" method="post" action="validpage.html"
onsubmit="return validateform()">
            Email: <input type="text" size="20" name="email"
onblur="emailvalidation()"><br>
            User Name: <input type="text" name="name"><br>
            Password: <input type="password" name="password"><br>
            <input type="submit" value="Submit" >
        </form>
    </body>
</html>
```


validpage.html //File name

```
<html>
    <body>
        <script type="text/javascript">
            alert("You are a Valid User
            !!!");
        </script>
    </body>
</html>
```

Output:





2.6: Event

Javascript | Error and Exceptional Handling

An error is an action which is inaccurate or incorrect.

There are three types of error in programming

1. Syntax error
2. Logical error
3. Runtime error

Syntax error:

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.

Logical error:

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.

Runtime Error:

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.

What is an Exception?

- An exception signifies the presence of an abnormal condition which requires special operable techniques.
- In programming terms, an exception is the anomalous code that breaks the normal flow of the code. Such exceptions require specialized programming constructs for its execution.

What is Exception Handling?

- In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them.
- It also enables to handle the flow control of the code/program.
- For handling the code, various handlers are used that process the exception and execute the code.
- **For example**, the Division of a non-zero value with zero will result into infinity always, and it is an exception. Thus, with the help of exception handling, it can be executed and handled.

Error Object

When a runtime error occurs, it creates and throws an Error object. Such an object can be used as a base for the user-defined exceptions too. An error object has two properties:

1. **name:** This is an object property that sets or returns an error name.
2. **message:** This property returns an error message in the string form.

Although Error is a generic constructor, there are following standard built-in error types or error constructors beside it:

1. **EvalError:** It creates an instance for the error that occurred in the eval(), which is a global function used for evaluating the js string code.
2. **InternalError:** It creates an instance when the js engine throws an internal error.
3. **RangeError:** It creates an instance for the error that occurs when a numeric variable or parameter is out of its valid range.
4. **ReferenceError:** It creates an instance for the error that occurs when an invalid reference is de-referenced.
5. **SyntaxError:** An instance is created for the syntax error that may occur while parsing the eval().
6. **TypeError:** When a variable is not a valid type, an instance is created for such an error.
7. **URIError:** An instance is created for the error that occurs when invalid parameters are passed in **encodeURI()** or **decodeURI()**.

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.

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

You can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

Here is the **try...catch...finally** block syntax –

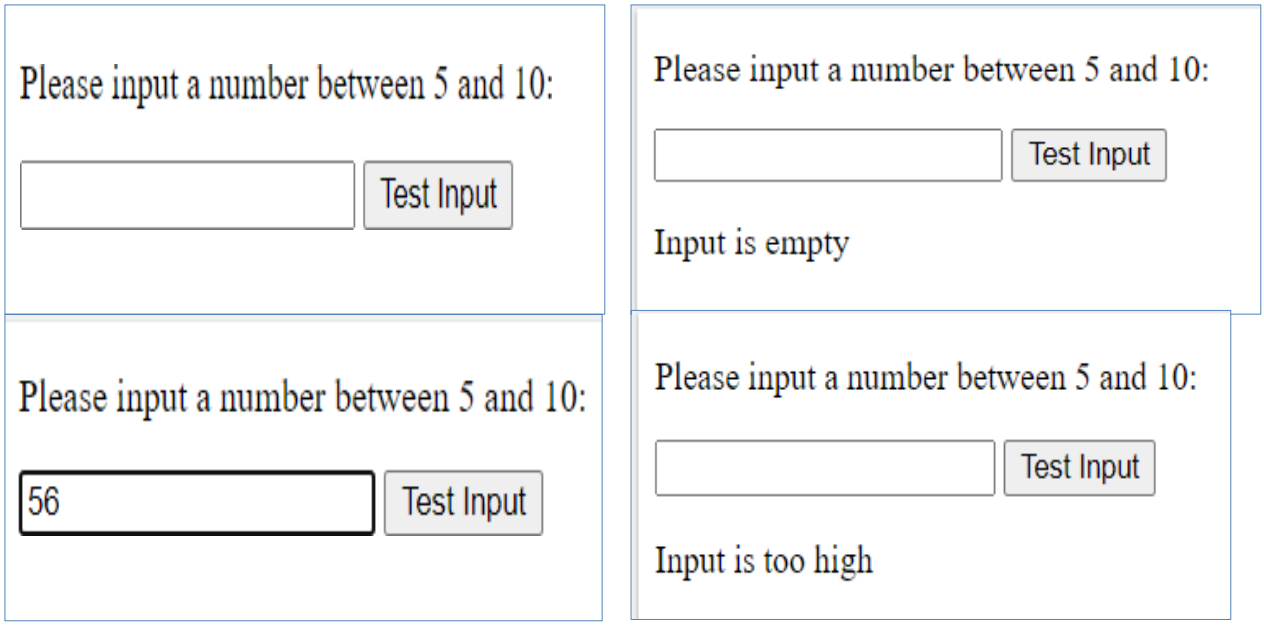
2.7: JavaScript Exception Handling

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.

Example 1:

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

```
<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>

<p id="p01"></p>

<script>
function myFunction()
  { var message, x;
  message = document.getElementById("p01");
  message.innerHTML = "";
  x  =  document.getElementById("demo").value;
  try {
    if(x == "") throw "is empty";
    if(isNaN(x)) throw "is not a
    number"; x = Number(x);
    if(x > 10)  throw "is too
    high"; if(x < 5) throw "is too
    low";
  }
  catch(err) {
    message.innerHTML = "Input " + err;
  }
  finally     {
    document.getElementById("demo").value = "";
  }
}
</script>

</body>
</html>
```



Example 2: [Type Error]

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Errors</h2>
<p>You cannot convert a number to upper case:</p>
<p id="demo"></p>

<script>
var num = 1;
try {
  num.toUpperCase();
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
</script>

</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:

```
<script type = "text/javascript">
  <!--
    try {
      /
      /

      C
      o
      d
      e

      t
      o
```

```
<html>
  <head>

    <script type = "text/javascript">
      <!--
      fu
        n
        c
```


```
    </script>
```

Built-in Objects

- Built-in objects are not related to any Window or DOM object model.
- These objects are used for simple data processing in the JavaScript.

Some of the built-in objects available in JavaScript are:

1) Date
2) Math
3) String, Number, Boolean
4) RegExp
5) window (Global Obejct)

1) Math Object

- Math object is a built-in static object.
- It is used for performing complex math operations.

Math Properties

Math Property	Description
SQRT2	Returns square root of 2.
PI	Returns Π value.
E \	Returns Euler's Constant.
LN2	Returns natural logarithm of 2.
LN10	Returns natural logarithm of 10.
LOG2E	Returns base 2 logarithm of E.
LOG10E	Returns 10 logarithm of E.

Math Methods

Methods	Description
abs()	Returns the absolute value of a number.
acos()	Returns the arccosine (in radians) of a number.
ceil()	Returns the smallest integer greater than or equal to a number.
cos()	Returns cosine of a number.
floor()	Returns the largest integer less than or equal to a number.
log()	Returns the natural logarithm (base E) of a number.

max()	Returns the largest of zero or more numbers.
min()	Returns the smallest of zero or more numbers.
pow()	Returns base to the exponent power, that is base exponent.

Example: Simple Program on Math Object Methods

```
<html>
    <head>
        <title>JavaScript Math Object Methods</title>
    </head>
    <body>
        <script type="text/javascript">

            var value = Math.abs(20);
            document.write("ABS Test Value : " + value +"<br>");

            var value = Math.acos(-1);
            document.write("ACOS Test Value : " + value +"<br>");

            var value = Math.asin(1);
            document.write("ASIN Test Value : " + value +"<br>");

            var value = Math.atan(.5);
            document.write("ATAN Test Value : " + value +"<br>");
        </script>
    </body>
</html>
```


Output

ABS Test Value : 20
ACOS Test Value : 3.141592653589793
ASIN Test Value : 1.5707963267948966
ATAN Test Value : 0.4636476090008061

Example: Simple Program on Math Object Properties

```
<html>
    <head>
        <title>JavaScript Math Object Properties</title>
    </head>
    <body>
        <script type="text/javascript">
            var value1 = Math.E
            document.write("E Value is :" + value1 + "<br>");

            var value2 = Math.LN2
            document.write("LN2 Value is :" + value2 + "<br>");

            var value3 = Math.LN10
            document.write("LN10 Value is :" + value3 + "<br>");

            var value4 = Math.PI
            document.write("PI Value is :" + value4 + "<br>");
        </script>
    </body>
</html>
```


Output:

E Value is :2.718281828459045
LN2 Value is
:0.6931471805599453 LN10 Value
is :2.302585092994046 PI Value is
:3.141592653589793

2) Date Object

- Date is a data type.
- Date object manipulates date and time.
- Date() constructor takes no arguments.
- Date object allows you to get and set the year, month, day, hour, minute, second and millisecond fields.

Syntax:

```
var variable_name = new Date();
```

Example:

```
var current_date = new Date();
```

Date Methods

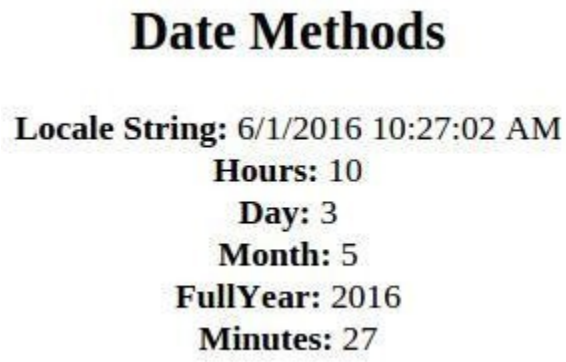
Methods	Description
Date()	Returns current date and time.
getDate()	Returns the day of the month.
getDay()	Returns the day of the week.
getFullYear()	Returns the year.
getHours()	Returns the hour.
getMinutes()	Returns the minutes.
getSeconds()	Returns the seconds.
getMilliseconds()	Returns the milliseconds.
getTime()	Returns the number of milliseconds since January 1, 1970 at 12:00 AM.
getTimezoneOffset()	Returns the timezone offset in minutes for the current locale.
getMonth()	Returns the month.
setDate()	Sets the day of the month.
setFullYear()	Sets the full year.
setHours()	Sets the hours.
setMinutes()	Sets the minutes.
setSeconds()	Sets the seconds.
setMilliseconds()	Sets the milliseconds.
setTime()	Sets the number of milliseconds since January 1, 1970 at 12:00 AM.
setMonth()	Sets the month.
toDateString()	Returns the date portion of the Date as a human-

	readable string.
toLocaleString()	Returns the Date object as a string.
toGMTString()	Returns the Date object as a string in GMT timezone.
valueOf()	Returns the primitive value of a Date object.

Example : JavaScript Date() Methods Program

```
<html>
    <body>
    <center>
        <h2>Date Methods</h2>
        <script type="text/javascript">
            var d = new Date();
            document.write("<b>Locale String:</b> " + d.toLocaleString()
+"<br>");
            document.write("<b>Hours:</b> " + d.getHours()+"<br>");
            document.write("<b>Day:</b> " + d.getDay()+"<br>");
            document.write("<b>Month:</b> " + d.getMonth()+"<br>");
            document.write("<b>FullYear:</b> " + d.getFullYear()+"<br>");
            document.write("<b>Minutes:</b> " + d.getMinutes()+"<br>");
        </script>
    </center>
    </body>
</html>
```

Output:



3) String Object

- String objects are used to work with text.
- It works with a series of characters.

Syntax:

```
var variable_name = new String(string);
```

Example:

```
var s = new String(string);
```

String Properties

Properties	Description
length	It returns the length of the string.
prototype	It allows you to add properties and methods to an object.
constructor	It returns the reference to the String function that created the object.

String Methods

Methods	Description
charAt()	It returns the character at the specified index.
charCodeAt()	It returns the ASCII code of the character at the specified position.
concat()	It combines the text of two strings and returns a new string.
indexOf()	It returns the index within the calling String object.
match()	It is used to match a regular expression against a string.
replace()	It is used to replace the matched substring with a new substring.
search()	It executes the search for a match between a regular expression.
slice()	It extracts a session of a string and returns a new string.

split()	It splits a string object into an array of strings by separating the string into the substrings.
toLowerCase()	It returns the calling string value converted lower case.
toUpperCase()	Returns the calling string value converted to uppercase.

Example : JavaScript String() Methods Program

```
<html>
    <body>
    <center>
        <script type="text/javascript">
            var str = "CareerRide Info";
            var s = str.split();
            document.write("<b>Char At:</b> " + str.charAt(1)+"<br>");
            document.write("<b>CharCode At:</b> " +
str.charCodeAt(2)+"<br>");
            document.write("<b>Index of:</b> " + str.indexOf("ide")+"<br>");
            document.write("<b>Lower Case:</b> " +
str.toLowerCase()+"<br>");
            document.write("<b>Upper Case:</b> " +
str.toUpperCase()+"<br>");
        </script>
    <center>
    </body>
</html>
```


Output:

Char At: a
CharCode At: 114
Index of: 7
Lower Case: careerride info
Upper Case: CAREERRIDE INFO

4) Boolean Object

- The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).

Syntax

Use the following syntax to create a **boolean** object.

```
var val = new Boolean(value);
```

Boolean Object Methods

Method	Description
toString()	Converts a Boolean value to a string, and returns the result
valueOf()	Returns the primitive value of a Boolean object

5) Number Object

- The Number object is an object wrapper for primitive numeric values. Number objects are created with new Number().

Syntax

```
var num = new Number(value);
```


Properties of Number object

- **Constructor** - Returns the function that created the Number object.
- **MAX VALUE** - Returns maximum numerical value possible in JavaScript.
- **MIN VALUE** - Returns minimum numerical value possible in JavaScript.
- **NEGATIVE INFINITY** - Represent the value of negative infinity.
- **POSITIVE INFINITY** - Represent the value of infinity.
- **Prototype** - Add properties and methods to an object.

Number Object Methods

Method	Description
toExponential(x)	Converts a number into an exponential notation
toFixed(x)	Formats a number with x numbers of digits after the decimal point

toPrecision(x)	Formats a number to x length
toString()	Converts a Number object to a string
valueOf()	Returns the primitive value of a Number object
toLocaleString() -	Returns a string value version of the current number in a format that may vary according to a browser's locale settings.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript toPrecision() Method </title>
  </head>
    <body>
    <script type = "text/javascript">
      var num = new Number(7.123456);

      document.write("Maximum Value " + Number.MAX_VALUE);
      document.write("<br />");
      document.write("Minimum Value " + Number.MIN_VALUE);
      document.write("<br />");
      document.write("num.toPrecision(4) is " + num.toPrecision(4));
      document.write("<br />");
      document.write("num.toExponential(4) is : " + num.toExponential(4));
      document.write("<br />");

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



```
Maximum Value 1.7976931348623157e+308
Minimum Value 5e-324
num.toPrecision(4) is 7.123
num.toExponential(4) is : 7.1235e+0
```

6) Window Object

- The window object represents an open window in a browser.
- If a document contain frames (<frame> or <iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

Window Object Methods

Method	Description
alert()	Displays an alert box with a message and an OK button
blur()	Removes focus from the current window
clearInterval()	Clears a timer set with setInterval()
clearTimeout()	Clears a timer set with setTimeout()
close()	Closes the current window
confirm()	Displays a dialog box with a message and an OK and a Cancel button
createPopup()	Creates a pop-up window
focus()	Sets focus to the current window
moveBy()	Moves a window relative to its current position
moveTo()	Moves a window to the specified position
open()	Opens a new browser window
print()	Prints the content of the current window
prompt()	Displays a dialog box that prompts the visitor for input
resizeBy()	Resizes the window by the specified pixels
resizeTo()	Resizes the window to the specified width and height
scroll()	
scrollBy()	Scrolls the content by the specified number of pixels
scrollTo()	Scrolls the content to the specified coordinates

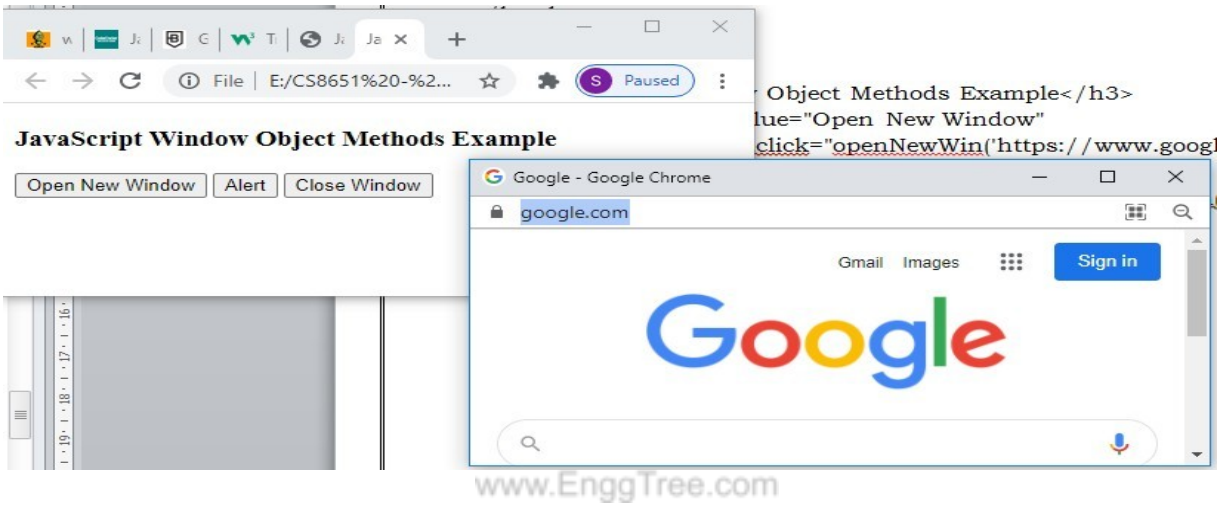
setInterval()	Calls a function or evaluates an expression at specified intervals (in milliseconds)
setTimeout()	Calls a function or evaluates an expression after a specified number of milliseconds

Example:

```
<!DOCTYPE HTML>
<html>
<head>
	<title>JavaScript Window Object Methods</title>
	<script type="text/javascript">
		var mywin;
		function openNewWin(url)
		{
			var wid =
			500; var hei =
			200;
			var winFeat = "width = " + wid + ", height = " + hei + " ,
status, resizable";
			myWin = window.open(url, "subWind", winFeat);
		}
		function disp_alert()
		{
			alert("Hi, This is an alert box.");
		}

		function close_win()
		{
			if(window.confirm("Do you really want to close the browser
?"))
				window.close();
		}
	</script>

</head>
```

```
<h3>JavaScript Window Object Methods Example</h3>
<input type="button" value="Open New Window"
                        onclick="openNewWin('https://www.google.com/');" />
<input type="button" value="Alert" onclick="disp_alert()" />
<input type="button" value="Close Window" onclick="close_win()" />

</body>
</html>
```

Output



Example : Simple Program on User-defined Function

```
<html>
<body>
    <script type="text/javascript">
    function add()          // Function Declaration
    {
        var a = 2,b =
        3; var sum =
        0; sum = a+b;
        document.write("<b>Addition: </b>"+sum);
    }
    </script>
    <p> Click the Button</p>
    <input type="button" onClick="add()" value="Click">          //add()
- Calling Function
```

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

Output:

Click the Button

Click

Addition: 5

2.8:

DHTML stands for **Dynamic Hypertext Markup language** i.e., **Dynamic HTML**.

Dynamic HTML is not a markup or programming language but it is a term that combines the features of various web development technologies for creating the web pages dynamic and interactive.

Components of Dynamic HTML

DHTML consists of the following four components or languages:

- HTML 4.0
- CSS
- JavaScript
- DOM.

HTML 4.0

HTML is a client-side markup language, which is a core component of the DHTML. It defines the structure of a web page with various defined basic elements or tags.

CSS

CSS stands for Cascading Style Sheet, which allows the web users or developers for controlling the style and layout of the HTML elements on the web pages.

JavaScript

JavaScript is a scripting language which is done on a client-side. The various browser supports JavaScript technology. DHTML uses the JavaScript technology for accessing, controlling, and manipulating the

HTML elements. The statements in JavaScript are the commands which tell the browser for performing an action.

DOM

DOM is the document object model. It is a w3c standard, which is a standard interface of programming for HTML. It is mainly used for defining the objects and properties of all elements in HTML.

Uses of DHTML

- ✓ It is used for designing the animated and interactive web pages that are developed in real-time.
- ✓ DHTML helps users by animating the text and images in their documents.
- ✓ It allows the authors for adding the effects on their pages.
- ✓ It also allows the page authors for including the drop-down menus or rollover buttons.
- ✓ This term is also used to create various browser-based action games.
- ✓ It is also used to add the ticker on various websites, which needs to refresh their content automatically.

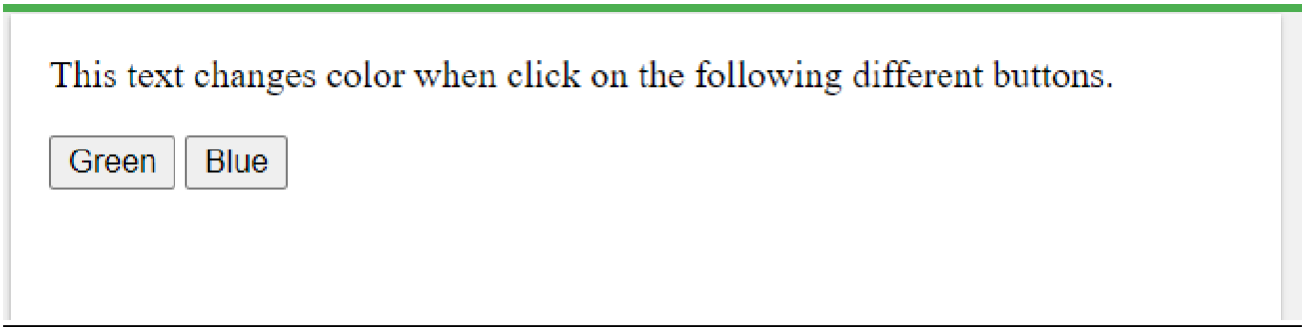
Difference between HTML and DHTML

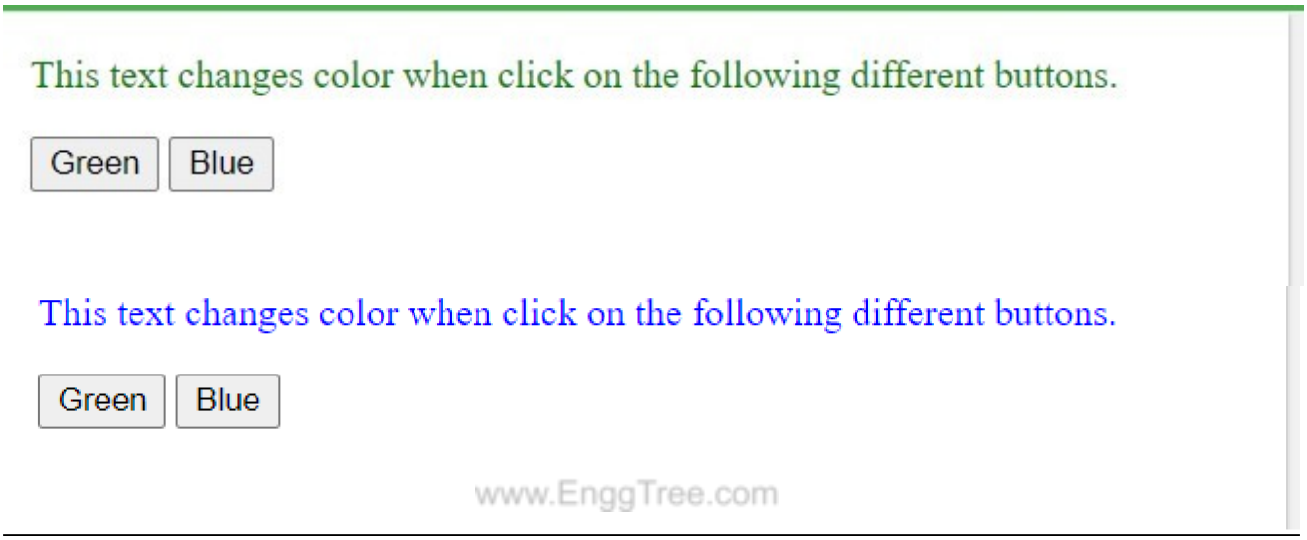
HTML (Hypertext Markup language)	DHTML (Dynamic Hypertext Markup language)
1. HTML is simply a markup language.	1. DHTML is not a language, but it is a set of technologies of web development.
2. It is used for developing and creating web pages.	2. It is used for creating and designing the animated and interactive web sites or pages.
3. This markup language creates static web pages.	3. This concept creates dynamic web pages.
4. It does not contain any server-side	4. It may contain the code of server-

scripting code.	side scripting.
5. The files of HTML are stored with the .html or .htm extension in a system.	5. The files of DHTML are stored with the .dhtm extension in a system.
6. A simple page which is created by a user without using the scripts or styles called as an HTML page.	6. A page which is created by a user using the HTML, CSS, DOM, and JavaScript technologies called a DHTML page.
7. This markup language does not need database connectivity.	7. This concept needs database connectivity because it interacts with users.

Example: Webpage using DHTML (HTML + DOM + CSS + JavaScript)

```
<html>
<head>
 <title>
changes the particular HTML element example
</title>
</head>

<body>
  <p id="demo"> This text changes color when click on the following different buttons. </p>
  <button onclick="change_Color('green');"> Green </button>
  <button onclick="change_Color('blue');"> Blue </button>
<script type="text/javascript">

function change_Color(newColor) {
  var element = document.getElementById('demo').style.color = newColor;
}
</script>
</body>
</html>
```





2.9: [illegible]

JSON stands for Javascript Object Notation. JSON is a text-based data format that is used to store and transfer data.

For example,

```
// JSON syntax
{
   "name": "John",
   "age": 22,
   "gender":
   "male",

}
```

In JSON, the data are in key/value pairs separated by a comma ,.

JSON was derived from JavaScript. So, the JSON syntax resembles JavaScript object literal syntax. However, the JSON format can be accessed and be created by other programming languages too.

2.10.1 : JSON Syntax

The JSON syntax is a subset of the JavaScript syntax.

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- ✓ Data is in name/value pairs.
- ✓ The name-value pairs are grouped by a colon (:) and separated by a comma (,)
- ✓ Data is separated by commas
- ✓ Curly braces hold objects
- ✓ Square brackets hold arrays
- ✓ An array begins with a left bracket and ends with a right bracket []
- ✓ Each key within the JSON should be unique and should be enclosed within the double quotes.
- ✓ The boolean type matches only two special values: true and false and NULL values are represented by the null literal (without quotes).

❖ **JSON Data - A Name and a Value**

- ✓ JSON data is written as name/value pairs.
- ✓ A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value.

❖ **JSON Values**

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

- a string
- a number
- an object (JSON object)
- an array

- a boolean
- null

In **JavaScript** values can be all of the above, plus any other valid JavaScript expression, including:

- a function
- a date
- undefined

Example "name"

❖ JSON Object

2.10: JSON introduction – Syntax – Function Files

The JSON object is written inside curly braces { }. JSON objects can contain multiple **key/value** pa

❖ JSON Array

JSON array is written inside square brackets [ ]. For example,

```
// JSON array
[ "apple", "mango", "banana"]

// JSON array containing objects
[
   { "name": "John", "age": 22 },
   { "name": "Peter", "age": 20 }.
   { "name": "Mark", "age": 23 }
]
```


❖ Accessing JSON Data

You can access JSON data using the dot notation. For example,

```
    "name":
    "John", "age":
    22,
    "hobby": {
          "reading" : true,
          "gaming" : false,
          "sport" :
          "football"
    },
    "class" : ["JavaScript", "HTML", "CSS"]
}

// accessing JSON object
console.log(data.name); //
John
console.log(data.hobby); // { gaming: false, reading: true, sport: "football"}
console.log(data.hobby.sport); // football
console.log(data.class[1]); // HTML
```


We use the. notation to access JSON data. Its syntax is:

```
variableName.key
```

You can also use square bracket syntaxto access JSON data. For ex

```
/
/

J
S
O
```

[]

Though the syntax of JSON is similar to the JavaScript object, JSON is different from JavaScript objects.

JSON	JavaScript Object
The key in key/value pair should be in double quotes.	The key in key/value pair can be without double quotes.
JSON cannot contain functions.	JavaScript objects can contain functions.
JSON can be created and used by other programming languages.	JavaScript objects can only be used in JavaScript.

JavaScript Objects VS JSON

Converting JSON to JavaScript Object

You can convert JSON data to a JavaScript object using the built-in function. For example,

```
// json object
const jsonData = '{ "name": "John", "age": 22 }';

// converting to JavaScript object const obj = JSON.parse(jsonData);

// accessing the data console.log(obj.name); // John
```


JSON.parse()

Converting JavaScript Object to JSON

You can also convert JavaScript object to JSON format using the JavaScript built-in function. For example,

```
// JavaScript object
const jsonData = { "name": "John", "age": 22 };

// converting to JSON
```

JSON.stringify()

2.10.2 : JSON Function Files

```
const obj = JSON.stringify(jsonData);

// accessing the data
console.log(obj); // "{"name":"John","age":22}"
```

- ✓ A common use of JSON is to read data from a web server, and display the data in a web page.
- ✓ This chapter will teach you, in 4 easy steps, how to read JSON data, using function files.

❖ JSON Example

This example reads a menu from **myTutorials.js**, and displays the menu in a web page:

JSON Example

```
<div id="id01"></div>

<script>
function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i<arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' + arr[i].display + '</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
```

```
}
</script>

<script src="myTutorials.js"></script>
```

Example Explained

1: Create an array of objects.

- ✓ Use an **array literal** to declare an **array** of **objects**.
- ✓ Give each object two properties: **display** and **url**.
- ✓ Name the array **myArray**:

myArray

```
var myArray = [
{
"display": "JavaScript Tutorial",
"url":  "https://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url":  "https://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url":  "https://www.w3schools.com/css/default.asp"
}
]
```

2: Create a JavaScript function to display the array.

Create a function **myFunction()** that loops the array objects, and display the content as HTML links:

myFunction()

```
function myFunction(arr) {
    var out = "";
```

```
    r i;
    for(i = 0; i < arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' + arr[i].display + '</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
}
```

Call **myFunction()** with **myArray** as argument:

❖ **Use of JSON**

```
myFunction(myArray);
```

3: Use an array literal as the argument (instead of the array variable):

Call **myFunction()** with an array **literal** as argument:

Calling myFunction()

```
myFunction([
{
"display": "JavaScript Tutorial",
"url": "https://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "https://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "https://www.w3schools.com/css/default.asp"
}
]);
```

4: Put the function in an external js file

Put the function in a file named **myTutorials.js**:

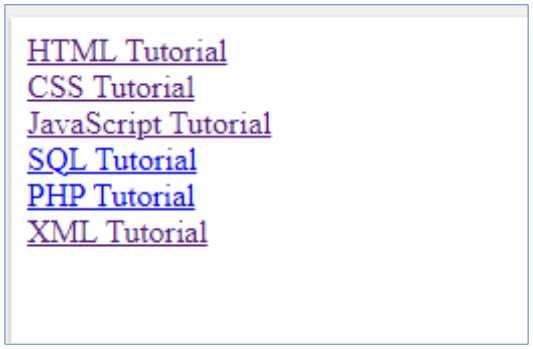
myTutorials.js

```
myFunction([
{
"display": "JavaScript Tutorial",
"url": "https://www.w3schools.com/js/default.asp"
},
{
"display": "HTML Tutorial",
"url": "https://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "https://www.w3schools.com/css/default.asp"
}
]);
```

Add the external script to your page (instead of the function call):

Add External Script

```
<script src="myTutorials.js"></script>
```

UNIT III SERVER SIDE PROGRAMMING

Servlets: Java Servlet Architecture- Servlet Life Cycle- Form GET and POST actions- Session Handling- Understanding Cookies- Web Server- DATABASE CONNECTIVITY:JDBC

3.1: An Introduction to Server Side Programming

SERVER-SIDE PROGRAMMING

Server-Side Programming means that writing scripts that are executed on the server and are then translated into HyperText Markup Language (HTML) which can be viewed by all web browsers.

Server-Side Programming Language:

A language used to develop programs that run on the server is called Server-Side Programming Language.

Difference between Server-Side Scripting and Client-Side Scripting:

S. No.	Client-Side Scripting	Server-Side Scripting
1.	Client-side scripting is used when the browser already has all the code & the page is altered on the basis of user input.	Server-side scripting is used to create dynamic pages based on browser request.
2.	Web browser executes the client-side scripting.	Web server executes the server-side scripting.
3.	The browser receives the page sent by the server & executes the client-side scripts.	Server executes server-side scripts to send out a page but it does not execute client-side scripts.
4.	Client-side scripting cannot be used to connect to the databases.	Server-side scripting is used to connect to the databases on the web server.

5.	Client-side scripting cannot access file system on the web server.	Server-side scripting can access the file system on the web server.
6.	Responses from client-side scripts are faster because the scripts are processed on local computer.	Responses from server-side scripts are slower because the scripts are processed on remote computer.
7.	Less Secure	More Secure
8.	Less Customization	High Customization
9.	More tasks at the browser	More tasks at the server
10.	Examples: JavaScript, VBScript, Dart etc.,	Examples: Servlets, JSP, PHP, ASP, ASP.net, Ruby, Perl, Python, ColdFusion.

3.2: Servlets

3.2.1 : What are Servlets?

Servlets are Java programs that run on a web server and act as a middle layer between a requests coming from a web browser or other HTTP client and databases or applications on the server.

Using servlets,

- ✓ We can collect input from users through web page forms, present records from a databases (or) another source and create web pages dynamically.
- ✓ We can develop sites with secure access, interact with DB, and maintain unique session info of each client.

Types of web server responses:

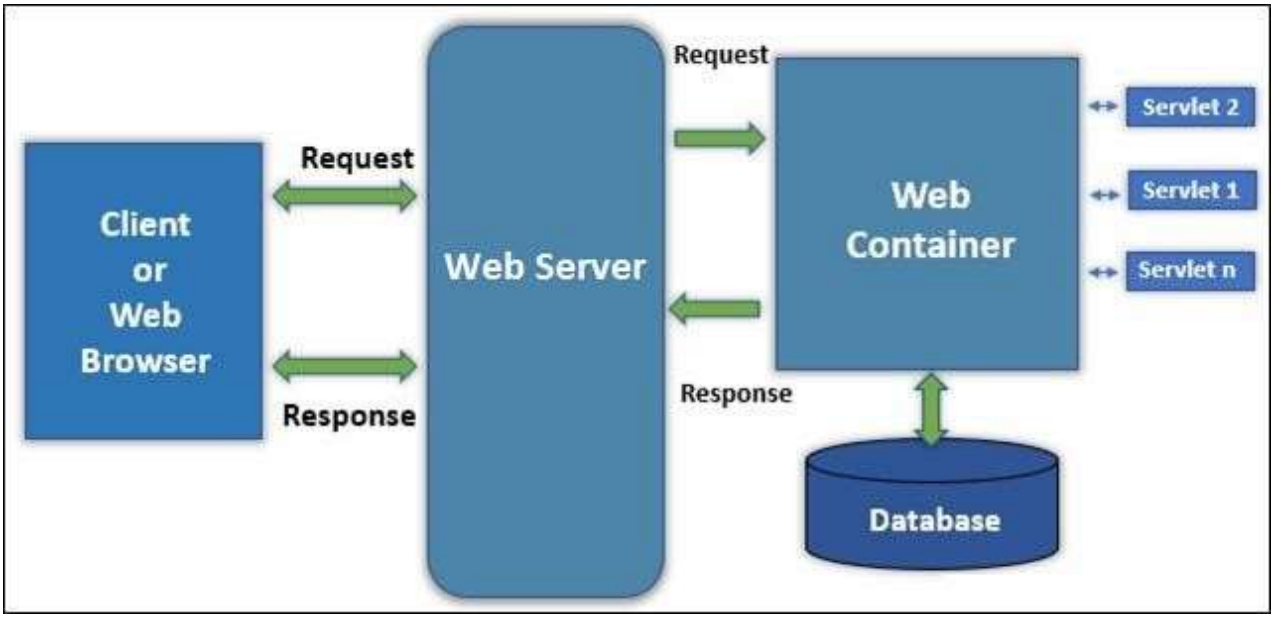
1. **Static Response:** HTML document retrieved from the file system and returned to the client is called static response.
2. **Dynamic Response:** HTML document is generated by a program in response to an HTTP request.

3.2.2 : Java Servlet Architecture

- ✓ Servlet architecture comes under a java programming language used to create dynamic web applications.
- ✓ Mainly servlets are used to develop server-side applications.
- ✓ Servlets are very robust and scalable.
- ✓ There are two types of Servlets-
  - 1.Generic Servlets
  - 2.HTTPServlets.
- ✓ Servlets can be created in three possible ways
  1) Implementing Servlet Interface
  2) Extending Generic Servlet.
  3) Extending HTTPServlet.
- ✓ Three life cycle methods available with servlets are init(), service() and destroy(). Every servlet should override these methods.

Components of Servlet Architecture

Below is the diagram to show how components working on servlet architecture.



1. Client

In this architecture, the web browser acts as a Client. Client or user connected with a web browser. The client is responsible for sending requests or HttpRequest to web server and processing responses received by the Web server.

2. Web Server

Web server controls how web user access hosted files and it is responsible for processing user request and responses. Here server is a software that understand URLs and HTTP protocol. Whenever browser needs to request file on the webserver, it process client request using HTTP request, if it finds requested file sends it back to browser through HTTP Response. There are two types' web servers Static and Dynamic webservers.in static web server sends the file as it is but in dynamic webserver hosted file is updated before it is sent to the browser.

3. Web Container

Web container is the component in the webserver it interacts with Java servlets. A web container is responsible for managing the lifecycle of servlets and it also performs the URL mapping task. Web container handles the requests of servlets, JSP and other files at the server-side. The important tasks performed by servlets are loading and unloading servlets, creating and managing requests and response objects and performs the overall tsk of servlet management.

Servlet Request Flow

Every servlet should override the following 3 methods namely:

1. init()
2. service()
3. destroy()

Following are the steps how servlet request has been processed consider the above diagram.

- The client sends a request.
- Web Server accepts the request and forwards it to the web container.
- Web container searches **web.xml** file for request URL pattern and gets the address of the servlet.

- If the servlet is not yet instantiated it will be instantiated and initialized by calling the **init()** method.
- The container calls public **service()** by passing ServletRequest and ServletResponse objects.
- Public **service()** method typecast ServletRequest and ServletResponse object to HttpServletRequest and HttpServletResponse objects respectively.
- Public service() method calls protected service().
- Protected service() method checks the client request & corresponding do_() method is called.
- Request is handled by sending the result generated by do_() to the client.

Uses of Servlet Architecture

Let us see some of the uses of servlet that are given below:

1. Servlets are used to form data manipulation like accepting form data and generating dynamic HTML pages.
2. Servlets helps in developing server load balancing applications. Where load balancing is among different servers.
3. Servlets are used as the middle tier in enterprise network platforms for connecting the SQL database.
4. Servlets can be integrated with applets to provide high-level interactivity and dynamic web content generation.
5. Servlet is used to develop applications where this act as an active agent in the middle tiers, where they share data with each other.
6. Since the servlet supports various protocols like HTTP, FTF, etc. this helps in developing applications like file server applications, chat enabled applications.

Advantages

Below are some important advantages of the servlet as follows:

- Servlets are server independent, as they are compatible with any web server. Compared to other server-side web technologies like ASP and JavaScript these are server-specific.
- Servlets are protocol-independent i.e. it supports FTP, SMTP, etc. Mainly it provides extended support to HTTP protocol functionality.

- Servlets are persistent because it remains in memory until explicitly destroyed this helps in several request processing and one database connection can handle several database requests.
- Servlets are portable since the servlets are written in java they are portable and supports any web server.
- Faster in execution, servlets are compiled into byte code execute more quickly compared to other scripting languages. Byte code conversion gives better performance and helps in type checking and error.

Disadvantages

- Designing a servlet can be pretty laborious.
- Exceptions need to be handled while designing a servlet since they are not thread-safe.
- Developers may need additional skills to program a servlet.

3.2.3 : Servlet Life Cycle

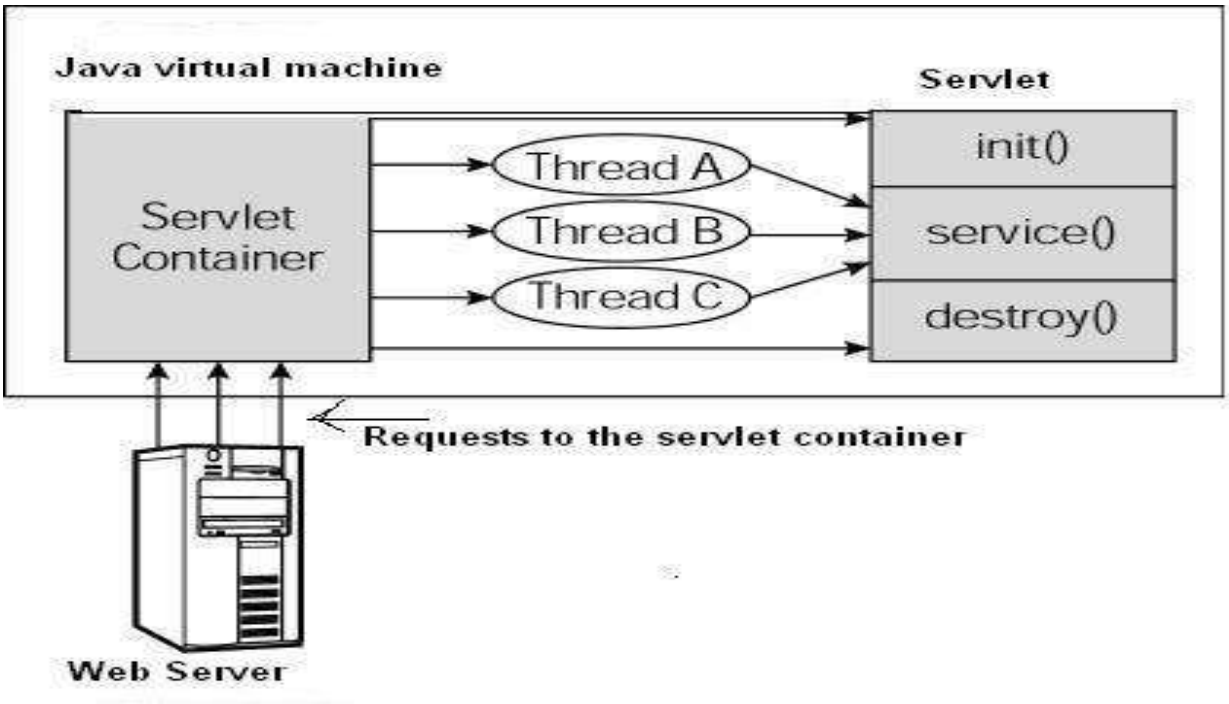
A servlet life cycle can be defined as the entire process from its creation till the destruction.

The following are the paths followed by a servlet:

1) The servlet is initialized by calling the init () method.
2) The servlet calls service() method to process a client's request.
3) The servlet is terminated by calling the destroy() method.
4) Finally, servlet is garbage collected by the garbage collector of the JVM.

Life cycle methods of a Servlet:

1) **init()**
2) **service()**
3) **destroy()**



1) The init() method:

This method is called when the servlet is first created. It is called only once during its lifecycle and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
public void init() throws ServletException {
// Initialization code...
}
```

2) The service() method :

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException{
}
```

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the cl ient.

The doGet() and doPost() are most frequently used methods with in each service request. Here are the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
// Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
// Servlet code
}

3) The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy() {
// Finalization code...
}
```

Structure of a servlet program

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;
public class NewServlet extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
response.setContentType("text/html"); // content type of the response PrintWriter out = response.getWriter(); // used to create a response as a Html doc try {
```

```
out.println("<html>");

out.println("</html>");

}catch(Exception  e){}
}
}
}
```

Servlets - Examples

Servlets are Java classes which service HTTP requests and implement the javax.servlet.Servlet interface.
Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Sample Code for Hello World:

Following is the sample source code structure of a servlet example to write Hello World:

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

// Extend HttpServlet class

public class HelloWorld extends HttpServlet { private String message;
public void init() throws ServletException
{
// Do required initialization
message = "Hello World";
}

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

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

// Actual logic goes here.
PrintWriter out =  response.getWriter();
out.println("<html><body><b>" + message + "</b></body></html>");
out.close();
}

public void destroy() { }
}
```

Output:

Finally type http://localhost:8080/HelloWorld in browser's address box. If everything goes fine, you would get following result:

3.2.4 : Parameter data and Query Strings

Servlet has methods to access data contained in HTTP Request (URL) sent to the server from the browser.

The Query String portion of the HTTP request is so called parameter data.

For example,

http://www.example.com/servlet/PrintThis?name=Raj&color=Red

where,

the portion after the ? is called a query string.

- ✓ Here it is "name=Raj&color=Red", in which name and color are parameter names and "Raj" and "Red" are parameter values.
- ✓ PrintThis is a servlet filename and
- ✓ servlet is a directory.
- ✓ Multiple parameters are separated by &.
- ✓ All parameter values are strings by default.
- ✓ Parameter names and values can be any 8-bit characters.

The following methods are used to process these parameter data in servlets.

TABLE 6.1: Some HttpServletRequest methods for accessing parameter data.

Method	Purpose
String getQueryString()	Returns the entire query string in its original (URL encoded) form.
Enumeration getParameterNames()	Returns Enumeration of String values representing all parameter names (URL decoded) in the query string.
String getParameter (String name)	Returns String representing value (URL decoded) of parameter named name, or null if parameter is not present in the query string.
String[] getParameterValues (String name)	Returns array of String's representing all values (URL decoded) of parameter named name, or null if parameter is not present in the query string.

The following program explains how to process these parameter names and values as well as path of the resource using servlet.

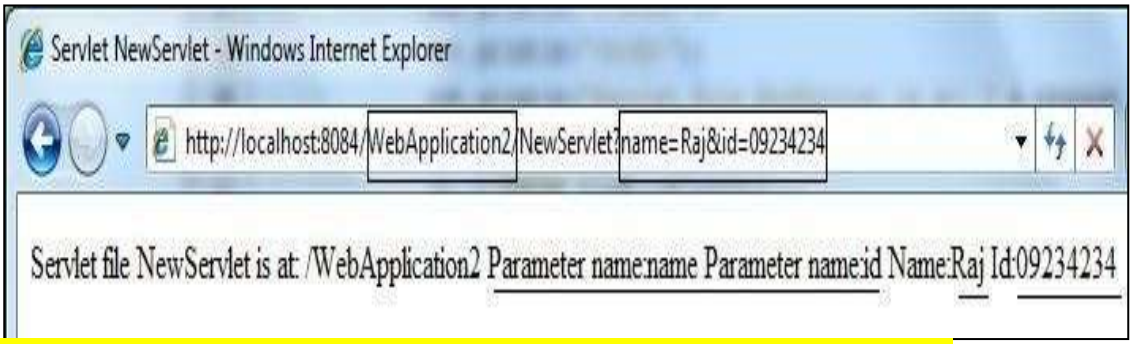
```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class NewServlet extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
try {
```

```
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet NewServlet</title>");
out.println("</head>");
out.println("<body>");
out.println("Servlet file NewServlet is at: " + request.getContextPath());
Enumeration para1=request.getParameterNames();
while(para1.hasMoreElements())
{
out.println("Parameter name:"+para1.nextElement());
}
String name = request.getParameter("name");
String id = request.getParameter("id");
out.println("Name:" + name);
out.println("Id:" + id);
out.println("</body>");
out.println("</html>");
}catch(Exception e){}
}
}
}
```


- ✓ The method getContextPath() of HttpServeltRequest object is used to get the location of the resource
- ✓ The method getParameter() is used to get the value of the parameter. The method getParameterNames() is used to return the paranameter names as well. It returns enumeration.
- ✓ The following code in the above program is used to retrieve the parameter names from the enumeration.

```
Enumeration para1=request.getParameterNames();
while(para1.hasMoreElements())
{
out.println("Parameter name:"+para1.nextElement());
}
```

Output:



3.2.5 : Form GET and POST Actions

Definition and Usage

The method attribute specifies how to send form-data (the form-data is sent to the page specified in the action attribute).
The form-data can be sent as URL variables (with method="get") or as HTTP post transaction (with method="post").

Notes on GET:

- It is used to process the query string which is part of URL
- Appends form-data into the URL in name/value pairs
- The length of a URL is limited (about 3000 characters)
- It is recommended when parameter data is not stored but used only to request information.
- Never use GET to send sensitive data! (will be visible in the URL)
- Useful for form submissions where a user wants to bookmark the result
- GET is better for non-secure data, like query strings in Google

Notes on POST:

- It is used to process the query string as well as to store the data on server.
- Appends form-data inside the body of the HTTP request (data is not shown in URL)
- Has no size limitations.
- It is recommended if parameter data is intended to cause the server to update stored data

- Most browsers will warn you if they are about to resubmit POST data to avoid duplicate updates
- Form submissions with POST cannot be bookmarked

BASIS FOR COMPARISON	GET	POST
Parameters are placed inside	URI	Body
Purpose	Retrieval of documents	Updation of data
Query results	Capable of being bookmarked.	Cannot be bookmarked.
Security	Vulnerable, as present in plaintext	Safer than GET method
Form data type constraints	Only ASCII characters are permitted.	No constraints, even binary data is permitted.
Form data length	Should be kept as minimum as possible.	Could lie in any range.
Visibility	Can be seen by anyone.	Doesn't display variables in URL.
Variable size	Up to 2000 character.	Up to 8 Mb
Caching	Method data can be cached.	Does not cache the data.

Example:

The following program explains how to send the data to server from a web page and the same how to receive it from the server.

Html for creating a web page

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

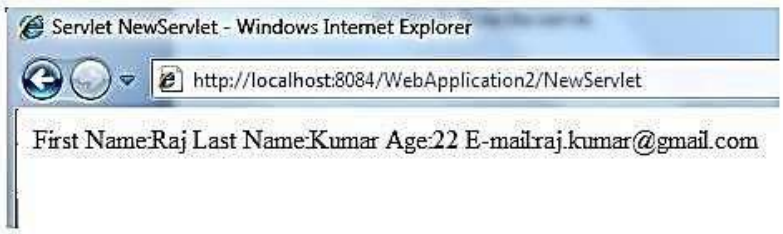
```
<body>
<pre>
<form action="NewServlet" method="post">
First Name: <input type="text" name="t1" />
Last Name: <input type="text" name="t2" />
Age: <input type="text" name="t3" />
E-mail:<input type="text" name="t4" />
<input type="submit" value="Submit" />
<form>
</pre>
</body>
</html>
```

Servlet for processing the data coming from this web page

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class NewServlet extends HttpServlet
{
public void goPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
try {
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet NewServlet</title>");
out.println("</head>");
out.println("<body>");
String s1 = request.getParameter("t1");
String s2 = request.getParameter("t2");
String s3 = request.getParameter("t3");
String s4 = request.getParameter("t4");
out.println("First Name:" + s1);
out.println("Last Name:" + s2);
```

```
out.println("Age:" + s3);
out.println("E-mail:" + s4);
out.println("</body>");
out.println("</html>");
} catch(Exception e) {}
}
}
```

Output



GET and POST methods in HTTP with examples.

❖ Handling GET request:

Login.html


```
<html>
<body>
<form action="login" method="get">
<table>
<tr>
<td>User</td>
<td><input name="user" /></td>
</tr>
<tr>
<td>password</td>
<td><input name="password" /></td>
</tr>
</table>
<input type="submit" />
</form>
</body>
</html>
```

create a Servlet which receives the request in /login , which is the indicated direction in the action attribute of the tag <form> of login.html

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet
{
protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
{
String user = req.getParameter("user");
String pass = req.getParameter("password");
if ("balamurugan".equals(user) && "bala1234".equals(pass))
{
response(resp, "login ok");
}
else
{
response(resp, "invalid login");
}
}
private void response(HttpServletResponse resp, String msg)
throws IOException
{
PrintWriter out = resp.getWriter();
out.println("<html><body>");
out.println(msg);
out.println("</body></html>");
}
}
```


We compilate this Servlet and we include LoginServlet.class in the folder /WEB-INF/classes. We modify web.xml to link /login with this Servlet.

web.xml

```
<web-app>
```

```
<servlet>
<servlet-name>hello</servlet-name>
<servlet-class> hello </servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>login-servlet</servlet-name>
<url-pattern>/login</url-pattern>
</servlet-mapping>
</web-app>
```

We restart the server, open the page login.html, write an "x" in user, write an "x" in password and click on the submit button.

Logging in with Wrong credentials

Logging in with right credentials

❖ Handling POST request:

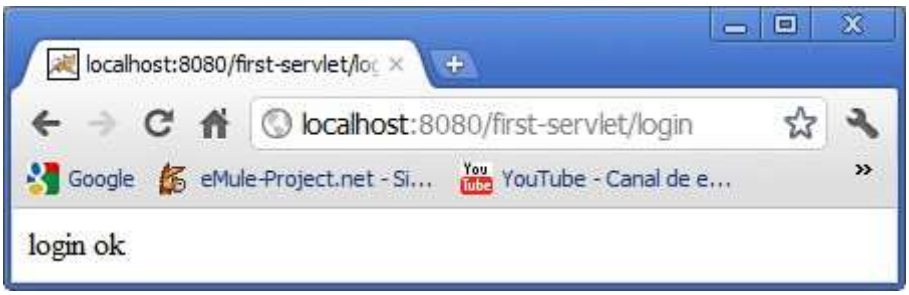
```
<html>
<body>
<form action="login" method="post">
<table>
<tr><td>User</td> <td><input name="user" /></td></tr>
<tr><td>password</td> <td><input name="password" /></td></tr>
</table>
<input type="submit" />
</form>
</body>
</html>
```

Reusing login.html, we will use the following error.

- What is happening here is that we haven´t implemented the doPost method (we have only implemented doGet), so our Servlet is not able to receive POST requests. In the following code we can see the necessary modifications to make it work.

```
import java.io.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet
{
protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException
{
String user = req.getParameter("user");
String pass = req.getParameter("password");
if ("balamurugan".equals(user) && "bala1234".equals(pass))
{
response(resp, "login ok");
}
else
{
response(resp, "invalid login");
}
}
private void response(HttpServletResponse resp, String msg)
throws IOException
{
PrintWriter out = resp.getWriter();
out.println("<html><body>");
out.println(msg);
out.println("</body></html>");
}
}
```

We can see that the parameters of the URL have disappeared.

3.2.6 : Session Handling

There are two types of protocols:

1. **Stateful Protocol:** In this protocol, part of data is exchanged between client and server and these protocols always keep track of communication sessions.
2. **Stateless protocol:** It is a protocol in which neither client nor server keeps track of the state of communication session.

- A **session** is simply the limited interval of time in which two systems communicate with each other.
- **Tracking** is the recording of the thing under session.
- Session Tracking is remembering and recording of client conversion in span of time.
- **Formal Definition:**
  Session Tracking is a technique by which we can keep track of previous sessions (conversations) held between server and the browser by using the session ID. It is also called as session management.
- **Session ID:** It is a unique identification number to identify HTTP requests. For sending all state information to and fro between the browser and server, session ID is used.

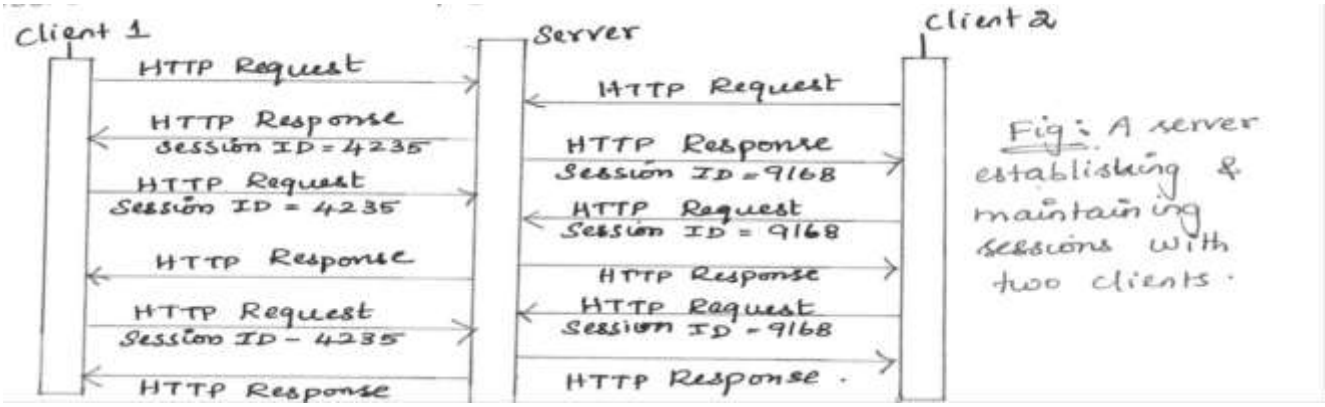


Fig: A server establishing & maintaining sessions with two clients.

- If web application is capable of remembering and recording of client conversion in span of time then that web application is called as **stateful web application.**

- **Why need/use Session Tracking?**

- Http protocol is stateless, that means, each time user requests to the server, server treats the request as the new request. So we use session tracking technique to keep track of previous conversations held between

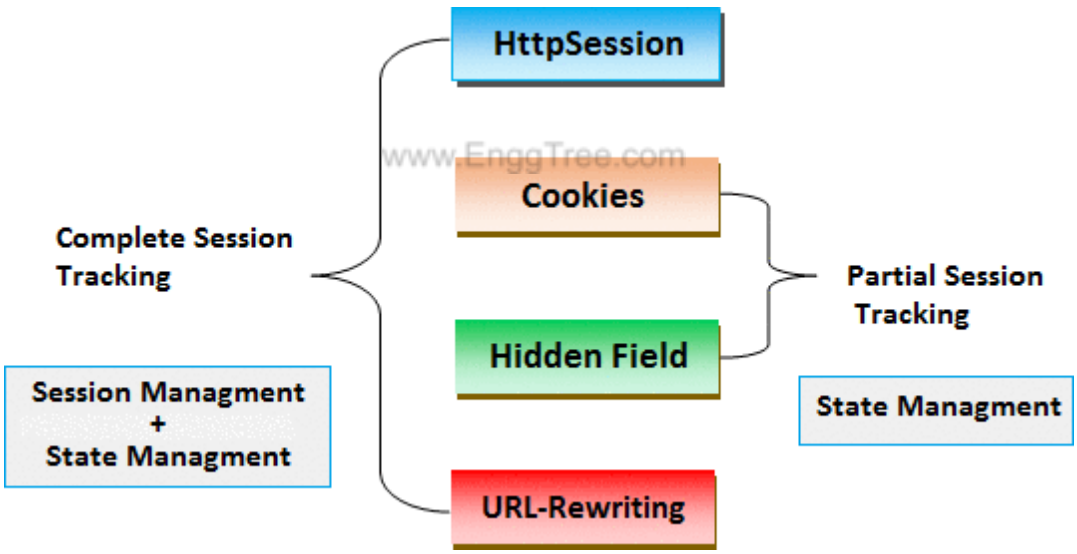
client and server and to maintain the state of a user to recognize a particular user.

- Session Tracking is useful for online shopping, mailing application, E-Commerce application to track the conversion.

Session Tracking Techniques

Servlet technology allows four techniques to track conversion, they are;

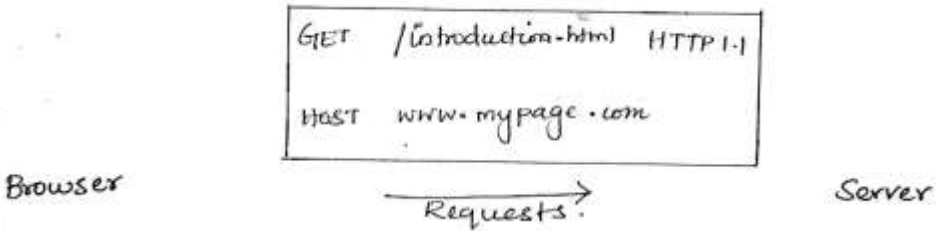
1) Cookies
2) URL Rewriting
3) Hidden Form Field
4) HttpSession



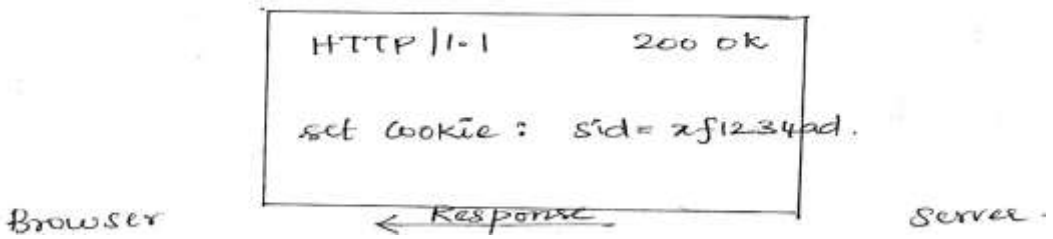
1. Cookies

- ✓ **A cookie is a name-value pair of information that a web server sends to a client machine as a part of an HTTP response.**
- ✓ The browser then returns these cookies unchanged to the server to indicate the state of the conversation.
- ✓ By returning a cookie to a web server, the browser provides the server a means of connecting the current page view with the previous page view.

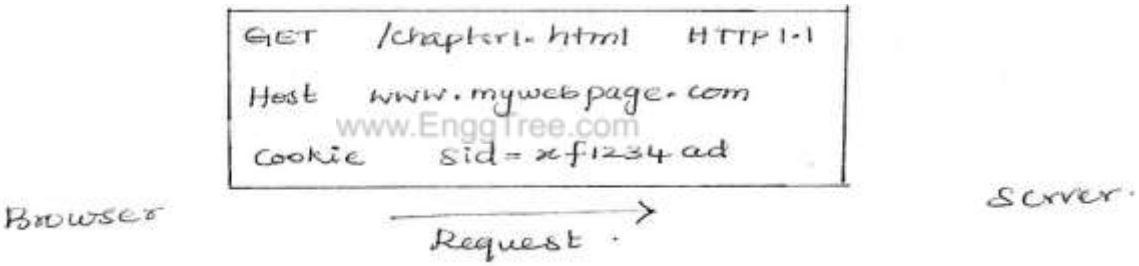
Example:



(i) Browser connects to the server www.mypage.com by making a request.



(ii) The server then replies in the form of HTTP response, session id is given in the form of cookie.



(iii) Another request to the same server. By including cookie which contain sid=xf1234ad server knows that this request is related to the previous one. Then server-browser can keep track of current session.

Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

1. Non-persistent cookie

It is valid for single session only. It is removed each time when user closes the browser.

2. Persistent cookie

It is valid for multiple session. It is not removed each time when user closes the browser. It is removed only if user logout or sign out.

Advantage of Cookie

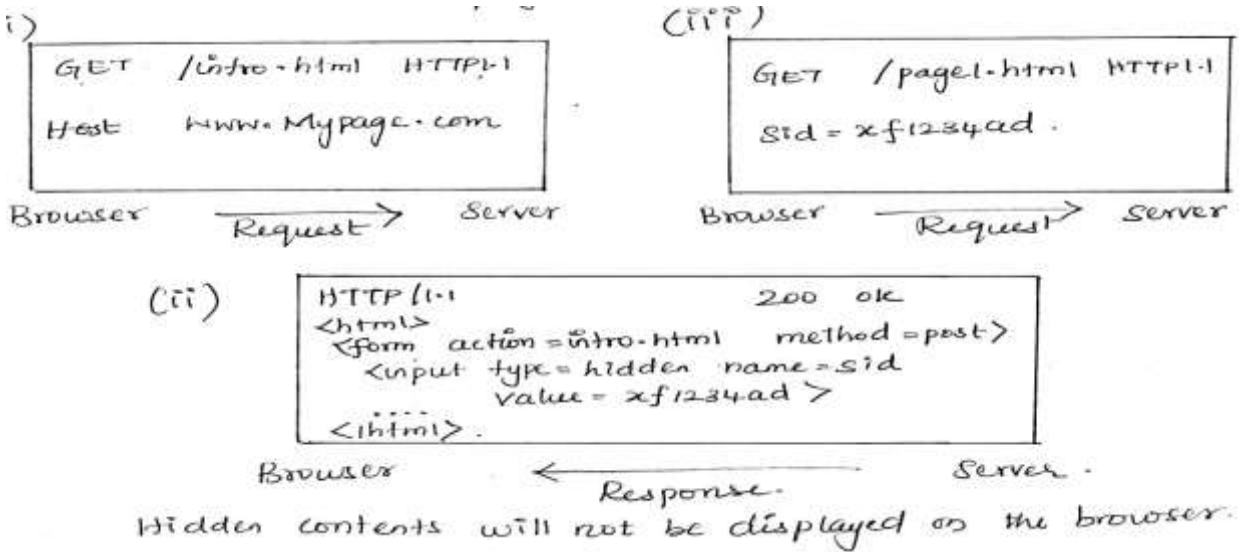
- ✓ Simplest technique of maintaining the state.
- ✓ Cookie are maintained at client side so they do not give any burden to server.
- ✓ All server side technology and all web server, application servers support cookies.
- ✓ Presistent cookies can remember client data during session and after session with expiry time.

Limitation of Coockie

- ❖ It will not work if cookie is disabled from the browser.
- ❖ Cookies are text files, It does not provides security. Anyone can change this file.
- ❖ With cookies need client support that means if client disable the coockies then it does not store the client location.
- ❖ Cookies cannot store java objects as values, they only store text or string.

2. Hidden Form Fields

- ✓ **Tracking client conversion using HTML hidden variables in secure manner is known as hidden form field.**
- ✓ In this method, when browser makes a request to the server for some web page, the server responds by inserting hidden form fields in the HTML page containing the session id and then sends the page to the browser.



Hidden contents will not be displayed on the browser.

- ✓ Each time when web browser sends request back, then session_id value can be used to keep track of the different web browsers.
- ✓ This could be an effective way of keeping track of the session but clicking on a regular (<a href=….>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

Hidden Form Field Advantage

- Basic knowledge of html is enough to work with this technique.
- It will always work whether cookie is disabled or not.
- Hidden boxes resides in web pages of browser window so they do not provide burden to the server.
- This technique can be used along with all kind of web server or application server.

Hidden Form Field Dis-Advantage

- More complex than URL Rewriting.
- It is maintained at server side.
- Extra form submission is required on each pages.
- Hidden form field can not store java object as values. They only store text value

- It Also increase network traffic because hidden boxes data travels over the network along with request and response.
- Hidden boxes does not provides data security because their data can be view through view source option.

Example of session tracking by using Hidden Form Field

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
response){
        try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");
        out.print("Welcome  "+n);

        //creating form that have invisible textfield
        out.print("<form action='servlet2'>");
        out.print("<input type='hidden' name='uname' value='"+n+"'>");
        out.print("<input type='submit' value='continue'>");
        out.print("</form>");
        out.close();

          }
                catch(Exception e){System.out.println(e);}
    }
}
```


SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
response)
        try{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
```

```
        //Getting the value from the hidden field
        String  n=request.getParameter("uname");
        out.print("Hello "+n);

        out.close();
        }
                catch(Exception e){System.out.println(e);}
    }
}
```

web.xml

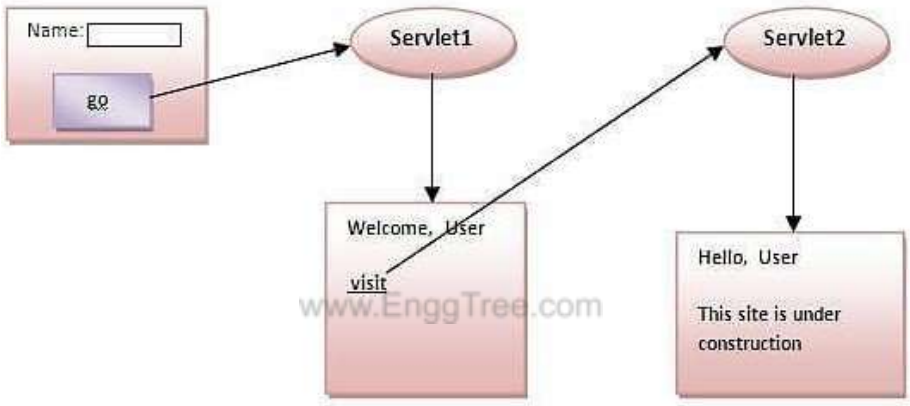
```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>


<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

3. URL Rewriting

- ✓ In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

- ✓ A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.



Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send only textual information.

Example of using URL Rewriting

In this example, we are maintaining the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

index.html

1. <form action="servlet1">

```
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class FirstServlet extends HttpServlet {
6.
7. public void doGet(HttpServletRequest request, HttpServletResponse respon
   se){
8.         try{
9.
10.                response.setContentType("text/html");
11.                PrintWriter out = response.getWriter();
12.
13.                String n=request.getParameter("userName");
14.                out.print("Welcome "+n);
15.
16.                //appending the username in the query string
17.                out.print("<a href='servlet2?uname="+n+"'>visit</a>");
18.
19.                out.close();
20.
21.                      }catch(Exception e){System.out.println(e);}
22.            }
23.
24.        }
```


SecondServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
```

```
4.
5. public class SecondServlet extends HttpServlet {
6.
7. public void doGet(HttpServletRequest request, HttpServletResponse respon
   se)
8.      try{
9.
10.             response.setContentType("text/html");
11.             PrintWriter out = response.getWriter();
12.
13.             //getting value from the query string
14.             String n=request.getParameter("uname");
15.             out.print("Hello "+n);
16.
17.             out.close();
18.
19.                 }catch(Exception e){System.out.println(e);}
20.         }
21.
22.
23.     }
```


web.xml

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-class>FirstServlet</servlet-class>
6. </servlet>
7.
8. <servlet-mapping>
9. <servlet-name>s1</servlet-name>
10.     <url-pattern>/servlet1</url-pattern>
11.     </servlet-mapping>
12.
13.     <servlet>
14.     <servlet-name>s2</servlet-name>
```

```
15.     <servlet-class>SecondServlet</servlet-class>
16.     </servlet>
17.
18.     <servlet-mapping>
19.     <servlet-name>s2</servlet-name>
20.     <url-pattern>/servlet2</url-pattern>
21.     </servlet-mapping>
22.
23.     </web-app>
```

4. HTTPSession Interface

An object of HttpSession can be used to perform two tasks:

1) bind objects
2) View and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

A. Creating a Session

- ✓ Java Servlet API supports the session concept by associating an HTTPSession object with each session maintained by the server.
- ✓ Each object stores the session ID for its session.
- ✓ HTTPSession object is created by the server when a servlet calls the **getSession()** method on its **HttpServletRequest** parameter and the associated HTTP request does not contain a valid session ID.
- ✓ The **getSession()** method returns the newly created object.
- ✓ If the HTTP request contains a valid session ID, then a call to **getSession()** returns the previously created HTTPSession object containing this session ID.

B. Storing and Retrieving Session Attributes:

A session attribute is a name-value pair that is stored in the HttpSession object.

The methods used to store and retrieve attributes are:

i) **setAttribute(String name, Object obj)**
Create or overwrite an attribute having the given name with given object value.

ii) **getAttribute(String name)**
Returns the value of the specified attribute (or) null if there is no attribute in HttpSession object.

iii) **removeAttribute(String value)**
Removes the specified attribute from HttpSession object.

iv) **getAttributeName()**
Returns the array of all attribute names associated with the HttpSession object.

C. Session Termination:

By default, each session expires if a server determines length of time elapses between a session's HttpRequest. Then the server destroys the corresponding session object.

Methods Used:

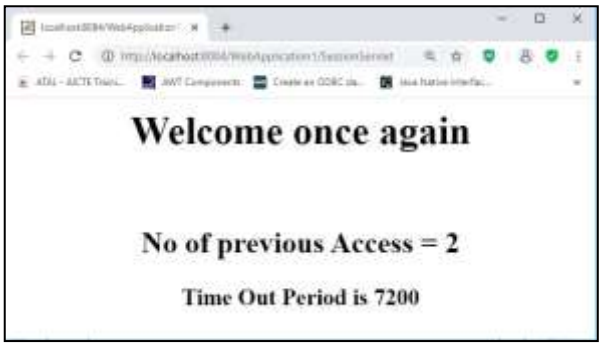
i) **setMaxInactiveInterval(int)**
This method takes an integer argument representing a number of seconds.
After the specified time elapses, the user's session will expire.

ii) **invalidate()**
If the timer completes before the next request containing the session ID is received, the server calls the invalidate() directly at any time in order to terminate the running session.

Example:

```
import java.io.*;
import  javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet(urlPatterns = {"/SessionServlet"})
public class SessionServlet extends HttpServlet {
   @Override
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException
    {
       response.setContentType("text/html");
       PrintWriter out=response.getWriter();
       HttpSession session=request.getSession();
       String heading;
       Integer count=(Integer)session.getAttribute("cnt");
       if(count==null)
       {
          count=new Integer(0);
          heading="Welcome!Thisis your first Access";
       }
       else
       {
          heading="Welcome once again";
          count=new Integer(count.intValue()+1);
       }
       session.setAttribute("cnt",count);
       out.println("<html><body><center>");
       out.println("<h1>"+heading+"</h1><br>");
       out.println("<h2> No of previous Access = "+count+"</h2>");
       session.setMaxInactiveInterval(2*60*60);
       out.println("<h3> Time Out Period is "+session.getMaxInactiveInterval());
       out.println("</h3></body></html>");
     }
}
```



3.2.7 : Understanding Cookies

Q6:

5. Cookies (X) (33) (27)

A cookie is a name-value pair (small information) that a web server sends to a client machine as part of an HTTP Response.

⇒ Browsers will store the cookie pair in a file on the client machine.

⇒ Then, before sending a request to a web server, the browser will check to see if it has stored any cookies received from this server.

⇒ If so, the browser will include these cookies in the Cookie header field of its HTTP request.

Cookie mechanism is a way to implement the session concept automatically as part of the processing performed by the getSession() method.

*) Cookie class:

→ Servlets can also explicitly create and use cookies.
→ Java Servlet API provides a class called cookie.
→ Each instance of this class corresponds to a single cookie.

⇒ Methods of Cookie class:

1) Cookie (String name, String value)	Constructs a cookie with given name and value.
2) String getName()	Returns the name of the cookie.
3) String getValue()	Returns the value of the cookie.
4) void setMaxAge (int expiry)	Sets the maximum age of the cookie in seconds for expiry.

(34) (28)

⇒ request.getCookies() – returns an array of cookie instances representing cookie data in Http request.

⇒ response.addCookie(cookie) – adds a cookie to the HTTP response.

Use of Cookies: Commonly used for session management.

⇒ Cookie Expiration:

*) Cookies can also expire like sessions, but the expiration is performed by the client, not by the server.

*) A ~~browser~~ By default, a cookie expires when the browser that accepted the cookie is ~~closed~~ closed.

*) But the server can request that, a cookie should expire after some time period using the method setMaxAge().

~~Examples~~

Uses of Cookies: (Advantages).

(i) used for storing session state.

(ii) Simplest technique for maintaining the session.

(iii) Cookies are maintained ~~at~~ client side at client side.

(iv) Used for website personalization.

Disadvantages:

(i) It will not work if cookie is disabled from the browser

(ii) Only textual information can be sent in Cookie object.

Example:

MyCookie.html

```
<html>
<head> <title> Cookie demo </title> </head>
<body>
<form name = "form1" method = "post"
    action = http://localhost:8080/examples/myServlet" >
<h3> Enter the Value for Cookie : </h3>
<input type = "text" name = "txt_data" >
<input type = "Submit" value = "Submit">
</form>
</body> </html>.
```

myServlet.java

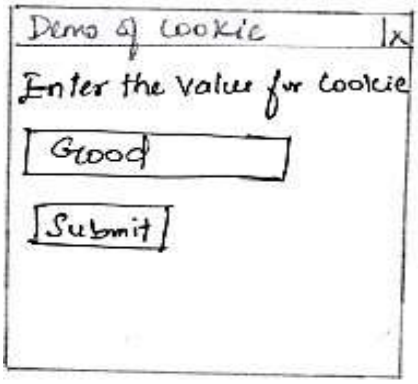
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class myServlet extends HttpServlet
{
public void doPost (HttpServletRequest req,
    HttpServletResponse res) throws ServletException, IOException
{
String txt_data = req.getParameter("txt_data");
Cookie cookie = new Cookie("My_cookie", txt_data);
res.addCookie(cookie);
res.setContentType("text/html");
PrintWriter out = res.getWriter();
out.println("<center> Cookie Added to the response </center>");
out.close())
}
}
```

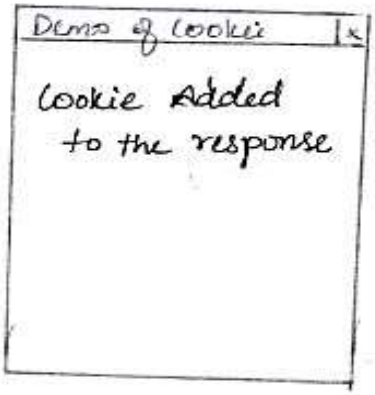
Reading cookies:

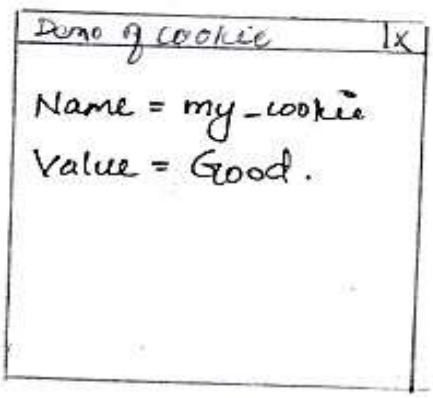
(36) (30)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
Public class getCookie extends HttpServlet
{
  public void doGet(HttpServletRequest req, HttpServletResponse
      res) throws ServletException, IOException
  {
    Cookie[] my_cookies = req.getCookies();
    res.setContentType("text/html");
    PrintWriter out = res.getWriter(); out.println("<b>");
    int n = my_cookies.length;
    for (int i=0; i<n; i++)
  {
    String name = my_cookies[i].getName();
    String value = my_cookies[i].getValue();
    out.println("name = " + name);
    out.println("Value = " + value);
  }
  out.close();
}
}
```

Output:







3.2.8 : Database Connectivity

Q7: (37) (31)

6. JDBC (JAVA DATABASE CONNECTIVITY)

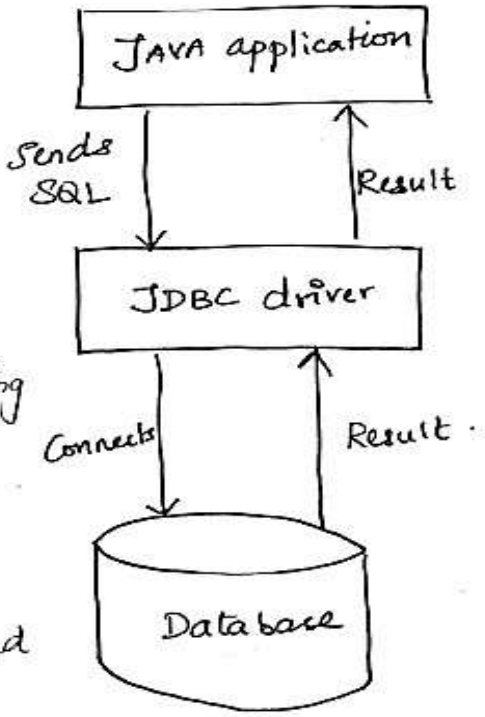
Definition: JDBC stands for Java Database Connectivity. JDBC is an API that consists of various classes, interfaces, exceptions using which Java application can send SQL statements to database.

*) The SQL (Structured Query Language) is used for accessing the database.

⇒ **HOW JDBC WORKS?**

Following steps illustrates the working of JDBC:

1) At first, Java application establishes connection with the data source.
2) Then Java application invokes classes and interfaces from JDBC driver for sending queries to the data source.
3) The JDBC driver connects to corresponding database & retrieves the result.
4) These results are based on SQL statements which are then returned to Java application.
5) Java application then uses the retrieved information for further processing.



⇒ **DIFFERENCE BETWEEN JDBC & ODBC.**

ODBC	JDBC.
1) Open DataBase Connectivity	Java DataBase Connectivity
2) Used for managing any kind of databases.	Specially created to support the Java programs.

(38) (39)

⇒ Uses of JDBC:

1) JDBC helps the java application to store and retrieve the data to and from the database.

2) JDBC allows the client to update the databases.

⇒ JDBC Architecture:

The JDBC architecture consists of two major components:

(1) JDBC API (2) JDBC Driver.

1. JDBC API:

The JDBC API is a set of classes, interfaces and exceptions used by an java application to establish connection with the database.

*) JDBC API is defined in the java.sql & javax.sql packages.

*) Core JDBC classes & Interfaces:

1) Driver Manager: Driver Manager class loads JDBC drivers in the memory. The Driver Manager also attempts to open a connection with the desired database.

Following statement is used to obtain the JDBC-ODBC bridge.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

2) Connection: This is an interface which represents the connectivity with the datasource. It is also used to create the statement instances.

```
Connection conn = DriverManager.getConnection(
        "jdbc:odbc:DSN", "username", "password");
```

3) **Statement:** This interface is used to represent and execute SQL statements. The following methods are available in this interface:
   - → execute() – create, delete commands
   - → executeQuery() – select command
   - → executeUpdate() – insert, update commands.

Syntax:

```
Statement stmt = conn.createStatement();
stmt.executeQuery("SELECT * FROM tableName");
```

4) **ResultSet:** It is an interface used to represent the result set retrieved from the database. We can navigate through the result set using the instance of ResultSet interface.

5) **SQLException:** Used to handle all SQL exceptions.

⇒ **JDBC Driver Types:**

There are four types of JDBC drivers:

1) **Type 1 driver: JDBC – ODBC bridge**

It connects JDBC software on the client to ODBC software. It is very slowest of all the drivers.

2) **Type 2 driver: Partial Java driver:**

It use native database API for data access.

3) **Type 3 driver: Pure Java driver for accessing middleware server**

It connects to the middle level server which in turn invokes the database.

4) Type 4 driver : Pure Java driver for direct access to database. (34)

It is loaded onto the client machine and supports direct communication with a DBMS. It does not need any supporting software other than the DBMS.

6.1 : SERVLET & DATABASE CONNECTIVITY USING JDBC.

[What is multi-tier application ? What is the need of it ?]

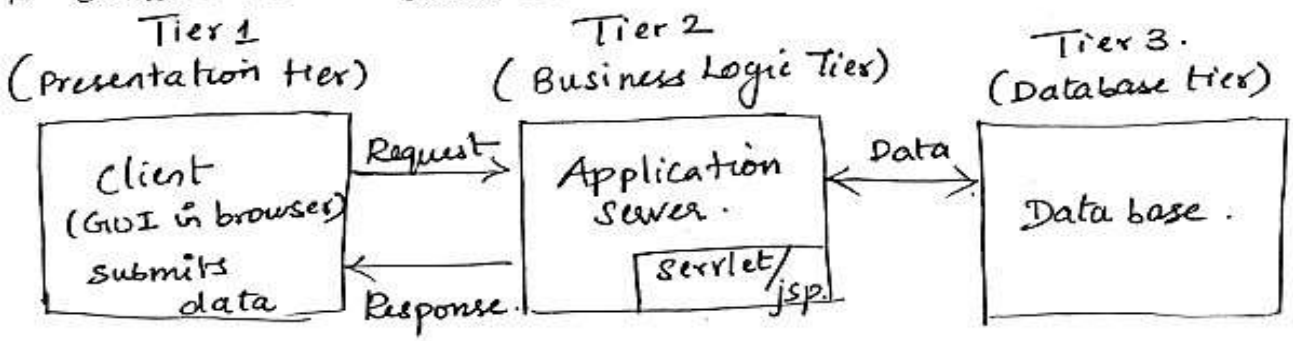
Ans:

A multi-tier architecture is basically a client-server architecture in which the application is to be executed by more than one programe.

⇒ In multi-tier architecture, the middle ware is a server to which the client submits some data and then the server sends this input data to the database.

⇒ Servlets used as a middle layer:

→ A Java servlet can be used as a business logic to connect to the databases.



In this architecture,

*) The presentation tier contains a program using which user can submit the request.
*) The request is then passed to application server which processes it.
*) Sometimes, external data may be required by the application server to process the request. Hence,
*) Databases are used as back end to supply the data required by application server.
*) Finally, application Server prepares the response object and request is fullfilled by presenting the requested resource to the user.

*) SERVLETS & JDBC CONNECTIVITY

Steps to connect Servlet to JDBC:

1) import java.sql.*; package is the servlet code.
2) Obtain the JDBC-Driver Manager to connect a servlet to the JDBC driver.

Syntax:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

3). Create an connection object to establish connection with the preferred database and to obtain the information about the database.

Syntax:

```
Connection conn = DriverManager.getConnection("jdbc:odbc:DSN",
                  "userName", "password");
```

(name of the database)

⇒ Steps to create DSN (Data Source Name)

(i) Open "Control Panel" and double click on "Administrative Tools".

(ii) Select the Tab "User DSN" (or) "System DSN" and click "Add" button.

(iii) From the list of drivers, select a driver for which you want to set up a data source.

(iv) Choose "Microsoft Access driver [.mdb, .accdb]" from the list. & click Finish button.

(v) In the window, type Data Source Name and select the database file for corresponding DSN. and click ok.

4) Create an object "stmt" of type statement which is used to send SQL statements to database and to retriere results produced by the statement (Query).

Syntax:

```
Statement stmt = conn.createStatement();
```

conn → Connection object.

5) Create an object of ResultSet. This objects stores the result set of records returned by stmt object.

*) It contains the methods for positioning to a certain row within the result set and for obtaining data from the fields in a row.

Syntax:

```
ResultSet rs = stmt.execute("SQL Query");
```

Example: JDBC program to list the contents of database table.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class TableDemo extends HttpServlet
{
public void service (HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
                                         IOException
{
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    PrintWriter out = response.getWriter();
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con = DriverManager.getConnection("jdbc:odbc:Dept","","");
stmt = con.createStatement();
rs = stmt.executeQuery("SELECT * FROM dept_table
        ORDER BY DNAME");
response.setContentType("text/html");
out.println("<html><head>");
out.println("<title> Servlet - DB Connectivity </title>");
out.println("<body><center>");
out.println("<h2> Department Details </h2>");
```

(44) (38)

```
out.println("<table border=3>");
out.print("<th> Dept_No </th>");
out.print("<th> Name </th>");
out.print("<th> Location </th>");
while(rs.next())
{
    out.print("<tr> <td>");
    out.print(rs.getString(1));
    out.print("</td> <td>");
    out.print(rs.getString(2));
    out.print("</td> <td>");
    out.print(rs.getString(3));
    out.print("</td></tr>");
}
out.print("</table></center></body></html>");
} catch(SQLException e)
{ }
```

Accessing fields in the row of resultset.


```
catch(ClassNotFoundException e1){e1.printStackTrace();}
catch(Exception e2){e2.printStackTrace();}.
finally { stmt.close(); rs.close(); con.close(); }.
} // service
} // class
```

Output :

Servlet-DB Connectivity

Department Details.

Dept_No	Name	Location
1001	Development	Delhi.
1002	Purchase	Pune
1003	Sales	Chennai

Database Schema:

dept_table :

Name	Type.
DNAME	Text
DNO	Number
DLOCATION	Text.

Program Explanation :

→ executeQuery() - used to execute the SQL statement

→ rs.next() - This method positions the result set counter at the next row each time it is called.

→ getString() – Used to retrieve the value stored in the fields in the row pointed to by the cursor.

→ sun.jdbc.odbc.JdbcOdbcDriver – Database driver to connect this java program with datasource.

→ jdbc:odbc:DSN. – Connects to the specific database.

UNIT IV PHP and XML

An introduction to PHP: PHP- Using PHP- Variables- Program control- Built-in functions- Form Validation- XML: Basic XML- Document Type Definition- XML Schema, XML Parsers and Validation, XSL.

4.1.1: INTRODUCTION TO PHP

What is PHP?

- ✓ PHP is an acronym for "PHP: Hypertext Preprocessor"
- ✓ PHP is a widely-used, open source server-side scripting language for creating dynamic web pages.
- ✓ PHP scripts are executed on the server.
- ✓ PHP is platform independent and free to download and use.

HISTORY:

- PHP was created by Rasmus Lerdorf to track users at his website.
- In 1995, Lerdorf released it as a package called the "Personal Home Page Tools".
- Two years later, PHP 2 features built-in database support and form handling.
- In 1997, PHP 3 was released with a rewritten parser, which substantially increased performance and led to an explosion of PHP use.
- The release of PHP 4 featured the new "Zend Engine" from Zend, a PHP software company. This version was faster and more powerful than its predecessor.
- Currently, PHP 5 features the new "Zend Engine 2" which provides further speed increases, exception handling and a new object-oriented programming model.

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code.
- PHP codes are executed on the server, and the result is returned to the browser as plain HTML.
- PHP files have extension ".php"

FEATURES OF PHP:

1. PHP can generate dynamic page content.
2. PHP can create, open, read, write, delete, and close files on the server.
3. PHP can collect form data.
4. PHP can send and receive cookies.
5. PHP can add, delete, and modify data in your database.

6. PHP can be used to control user-access.
7. PHP can encrypt data.
8. With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies.
9. You can also output any text, such as XHTML and XML.

ADVANTAGES OF PHP:

- ✓ Portability (Platform Independent) - PHP runs on various platforms (Windows, Linux, UNIX, Mac OS X, etc.)
- ✓ Performance - Scripts written in PHP executives faster than those written in other scripting language.
- ✓ Ease Of Use – It is easy to use. Its syntax is clear and consistent.
- ✓ Open Source - PHP is an open source project –it may be used without payment of licensing fees or investments in expensive hardware or software. This reduces software development costs without affecting either flexibility or reliability
- ✓ PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- ✓ PHP supports a wide range of databases.
- ✓ PHP is easy to learn and runs efficiently on the server side.
- ✓ Community Support - One of the nice things about a community-supported language like PHP is, accessing it offers to the creativity and imagination of hundreds of developers across the world.

INSTALLING PHP:

To program PHP, follow the below three steps:

- install a web server (Any Web server capable of executing PHP codes)
- install PHP
- install a database, such as MySQL

✓ The official PHP website (PHP.net) has installation instructions for PHP: http://php.net/manual/en/install.php

4.1.2: PHP – BASIC SYNTAX

BASIC PHP SYNTAX:

- ➢ A PHP script can be embedded directly and anywhere in the HTML document.
- ➢ A PHP script starts with <?php and ends with ?>:
- ➢ Syntax:

```
<?php
        // PHP code goes here
?>
```

- ➢ The default file extension for PHP files is ".php".
- ➢ A PHP file normally contains HTML tags, and some PHP scripting code.

- "echo" – statement used to output the text on a web page:

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

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

COMMENTS IN PHP:

- A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.
- Comments can be used to:
  - Let others understand what you are doing
  - Remind yourself of what you did.
- Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
// This is a single-line comment
# This is also a single-line comment
/*
This is a multiple-lines comment block that
 spans over Multiple lines
*/
$x = 5 /* + 15 */ + 5;
```

```
echo $x;
?>
</body>
</html>
```

PHP Case Sensitivity:

- In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive.
- However; all variable names are case-sensitive.
- Example:

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

<?php
// all the echo statements are treated as same
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";

// all the variables are treated as three different variables
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>

</body>
</html>
```



```
Hello World!
Hello World!
Hello World!

My car is red
My house is
My boat is
```

4.1.3: PHP – VARIABLES

Definition: Variables are "containers" for storing information.

Declaring PHP variables: In PHP, a variable starts with the $ sign, followed by the name of the variable.

Syntax: $variable_name

Initializing PHP variables: Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated or value on the right. The value of a variable is the value of its most recent assignment.

Syntax: $variable_name= expression or value;

Rules for PHP variables:

- ✓ A variable starts with the $ sign, followed by the name of the variable name.
- ✓ A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
- ✓ A variable name must start with a letter or the underscore character.
- ✓ A variable name cannot start with a number.
- ✓ A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _) but cannot use characters like + , - , % , (,) . &, etc.
- ✓ Variable names are case-sensitive ($age and $AGE are two different variables).

Example:

```
<html>
<head>
<title>Online PHP Script Execution</title>
</head>
<body>
<?php
   $number = 10;
   $text ="My variable contains the value of ";
   print($text.$number);
?>
</body>
</html>
```

Output:

My variable contains the value of 10

PHP VARIABLES SCOPE:

- ✓ In PHP, variables can be declared anywhere in the script.
- ✓ The scope of a variable is the part of the script where the variable can be referenced / used.
- ✓ PHP has three different variable scopes:
  - Local
  - Global
  - Static

❖ Global Scope:

A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function:

❖ Local Scope:

A variable declared within a function has a LOCAL SCOPE and can only be accessed within that function.

❖ PHP The static Keyword

- ✓ Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.
- ✓ To do this, use the static keyword when you first declare the variable:

Example:

```
<?php
$num1=1; // global variable
$x = 5; // global variable
$y = 10;    // global variable

function myTest() {
    $num1=11; // local variable
    static $z=0;
    $z++;
    echo "z = ".$z."<br>"; // static variable retains the last changed value
    global $x, $y;
    $y = $x + $y;
    echo "num1 inside function = ".$num1."<br>"; // prints 11
}
myTest();
mytest();
```

```
echo "x + y = ".$y."<br>"; // outputs 15
echo "num1 outside function= ".$num1; // prints 1
?>
```

```
z = 1
num1 inside function = 11
z = 2
num1 inside function = 11
x + y = 20
num1 outside function= 1
```

❖ **PHP The global Keyword**

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

- ✓ PHP also stores all global variables in an array called $GLOBALS[index].
- ✓ The index holds the name of the variable.
- ✓ **Example**


```
<?php
$x = 5;
$y = 10;

function myTest() {
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y; // outputs 15
?>
```

4.1.4: PHP – OUPUT STATEMENTS

❖ **PHP echo and print Statements:**

- ✓ echo and print are more or less the same. They are both used to output data to the screen.
- ✓ The differences are small:
  - o echo has no return value while print has a return value of 1 so it can be used in expressions.

- echo can take multiple parameters (although such usage is rare) while print can take one argument.
- echo is marginally faster than print.

✓ The echo statement can be used with or without parentheses: echo or echo().

✓ The print statement can be used with or without parentheses: print or print().

✓ **Example:**

```
<?php
$txt1 = "Learn PHP";
$txt2 = "deitel.com";
$x = 5;
$y = 4;

echo "<h2>$txt1</h2>";
print "Study PHP at " . $txt2 . "<br>";

echo("X + Y = ".($x + $y));
?>
```


Output:

Learn PHP
Study PHP at deitel.com
X + Y = 9

4.1.5: PHP – DATA TYPES

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

Type	Description
string	Text enclosed in either single („ „) or double (“ “) quotes.
int, Integer	Whole numbers (i.e., numbers without a decimal point)
float, double, real	Real numbers (i.e., numbers containing a decimal point)
bool, boolean	True or False
array	Group of elements

object	Group of associated data and methods
NULL	No value
resource	An external source – usually information from a database

❖ **PHP String:**
- ✓ A string is a sequence of characters, like "Hello world!".
- ✓ A string can be any text inside quotes. You can use single or double quotes.
- ✓ Example: $str1=”Hello World”; $str2=‟Hello World‟;

❖ **PHP Integer:**
- ✓ An integer is a whole number (without decimals).
- ✓ It is a number between -2,147,483,648 and +2,147,483,647.
- ✓ Rules for integers:
  - An integer must have at least one digit (0-9)
  - An integer cannot contain comma or blanks
  - An integer must not have a decimal point
  - An integer can be either positive or negative
  - Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)
- ✓ Example: $x=5;

❖ **PHP Float:**
- ✓ A float (floating point number) is a number with a decimal point or a number in exponential form.
- ✓ Example: $x=41.45;
- ✓ The PHP var_dump() function returns the data type and value.

❖ **PHP Boolean:**
- ✓ A Boolean represents two possible states: TRUE or FALSE.
- ✓ $x = true;
- ✓ $y = false;
- ✓ Booleans are often used in conditional testing.

❖ **PHP Array:**
- ✓ An array stores multiple values in one single variable.
- ✓ Example: $cars = array("Volvo","BMW","Toyota");

❖ **PHP Object:**
- ✓ An object is a data type which stores data and information on how to process that data.

- ✓ In PHP, an object must be explicitly declared.
- ✓ First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

❖ **PHP NULL Value:**

- ✓ Null is a special data type which can have only one value: NULL.
- ✓ A variable of data type NULL is a variable that has no value assigned to it.
- ✓ If a variable is created without a value, it is automatically assigned a value of NULL.
- ✓ Variables can also be emptied by setting the value to NULL:
- ✓ $x = null;

❖ **PHP Resource:**

- ✓ The special resource type is not an actual data type.
- ✓ It is the storing of a reference to functions and resources external to PHP.
- ✓ A common example of using the resource data type is a database call.

Example: Program using all the data types:

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


<?php
$x = 5985; // Integer variable
$y = 59.85; // float variable
$str1="Hello "; // string
$str2="World"; // string
$bool1=true; // boolean
$bool2=false; // boolean
$cars = array("Volvo","BMW","Toyota"); // array

class Car {     // class declartion
    function carModel() {     // function inside class
        $this->model = "VW";
        echo "Car Model from Object = ".$this->model."<br>";
    }
}
?>
<font color="blue" style="bold">

<?php
```

```
// create an object
$herbie = new Car(); // creating object for car class

// show object properties
echo $herbie->carModel();     //Invoking function through object

var_dump($x); echo "<br>";
var_dump($y); echo "<br>";
var_dump($str1); echo "<br>";
var_dump($str2); echo "<br>";
var_dump($bool1); echo "<br>";
var_dump($bool2); echo "<br>";
var_dump($cars); echo "<br>";
?>
</font>
</body>
</html>
```

Output:

```
Car Model from Object = VW
int(5985)
float(59.85)
string(6) "Hello "
string(5) "World"
bool(true)
bool(false)
array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }
```


CONVERSION BETWEEN DATA TYPES (TYPE COVERSION):

- ✓ Type Conversion is the process of converting one data type into another.
- ✓ Converting between different data types may be necessary when performing arithmetic operations with variables.
- ✓ Type conversions can be performed using the function settype.
- ✓ Syntax:

  settype($variable_name, "data_type");
- ✓ We can print the value of variables and their types using the function gettype.
- ✓ Syntax:

  gettype($variable_name);
- ✓ **Type Casting:**

  Another option for conversion between types is casting. Unlike settype, casting does not change a variable‟s content – it creates a

temporary copy of a variable‟s value in memory. Casting is useful when a different type is required in a specific operation but you would like to retain the variable‟s original value and type.

✓ **Example:**

```
<!DOCTYPE html>
<html>
<body>
   <?php

      $var1="114";
      $var2=114.12;
      $var3=114;

      print("$var1 is of ".gettype($var1)." type <br/>");
      print("$var2 is of ".gettype($var2)." type <br/>");
      print("$var3 is of ".gettype($var3)." type <br/>");

      echo("<b><u> Converting into other datatype </u></b><br/>");

      settype($var1,"double");
      settype($var2,"integer");
      settype($var3,"string");

      print("$var1 is of ".gettype($var1)." type <br/>");
      print("$var2 is of ".gettype($var2)." type <br/>");
      print("$var3 is of ".gettype($var3)." type <br/>");

      echo("<b><u> Type Casting </u></b><br/>");
      $data="98.6 degrees";
      print("<b>Before Casting : data is of ".gettype($data)." type </b><br/>");
      print("After Casting into double : value of data = ".(double)$data. "<br/>");
      print("After Casting into integer : value of data = ".(int)$data. "<br/>");
      print("<b>After Casting : data is of ".gettype($data)." type </b><br/>");

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

Result

114 is of string type
114.12 is of double type
114 is of integer type
Converting into other datatype
114 is of double type
114 is of integer type
114 is of string type
Type Casting
Before Casting : data is of string type
After Casting into double : value of data = 98.6
After Casting into integer : value of data = 98
After Casting : data is of string type

4.1.5: PHP – CONSTANTS

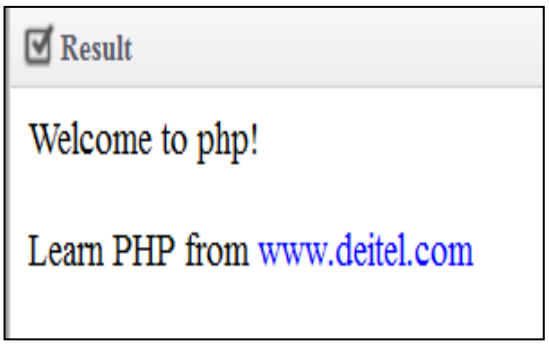
A constant is an identifier (name) for a simple value. The value cannot be changed during the script. Constants are like variables except that once they are defined they cannot be changed or undefined.

- ✓ A valid constant name starts with a letter or underscore (no $ sign before the constant name).
- ✓ **CREATING CONSTANTS:**
  - To create a constant, use the define() function.
  - Syntax:

    define (name, value, case-insensitive)
- ✓ Parameters:
  - name: Specifies the name of the constant
  - value: Specifies the value of the constant
  - Case-insensitive: Specifies whether the constant name should be case-insensitive. Default is false.
- ✓ Note: Unlike variables, constants are automatically global across the entire script. The example below uses a constant inside a function, even if it is defined outside the function.
- ✓ Example:

```
<html>
<head><title>Logical Operators</title><head>
<body>
<?php
define("GREETING", "Welcome to php!");

function test()
{
```

```
echo constant("GREETING"); echo "<br/><br/>";
define("TUTORIALS", "www.deitel.com",true);
echo "Learn PHP from <font color='blue'>".tutorials."</font>"; // no $
sign for constants
}
test();

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



DIFFERENCES BETWEEN CONSTANTS AND VARIABLES:

- ✓ There is no need to write a dollar sign ($) before a constant, where as in Variable one has to write a dollar sign.
- ✓ Constants cannot be defined by simple assignment, they may only be defined using the define() function.
- ✓ Constants may be defined and accessed anywhere without regard to variable scoping rules.
- ✓ Once the Constants have been set, may not be redefined or undefined.

4.1.6: PHP – OPERATORS

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

1) Arithmetic operators
2) Assignment operators
3) Comparison operators
4) Increment/Decrement operators
5) Logical operators
6) String operators
7) Array operators

❖ PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced in PHP 5.6)

❖ PHP Assignment Operators

✓ The PHP assignment operators are used with numeric values to write a value to a variable.

✓ The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

❖ PHP Comparison Operators

✓ The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result
= =	Equal	$x == $y	Returns true if $x is equal to $y
= = =	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!= =	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

❖ PHP Increment / Decrement Operators

✓ The PHP increment operators are used to increment a variable's value.
✓ The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

❖ PHP Logical Operators

✓ The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

❖ PHP String Operators

✓ PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

❖ PHP Array Operators

✓ The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

Example:

```
<html>
<head><title>Logical Operators</title><head>
```

```
<body>
<?php
   $x=10;
   $y=5;

   echo "<b><u> Arithmetic Operators </u></b><br/>";
   echo "Add = $x+$y = ".($x+$y)." <br>";
   echo "Sub = $x-$y = ".($x-$y)." <br>";
   echo "Mul = $x*$y = ".($x*$y)." <br>";
   echo "Div = $x/$y = ".($x/$y)." <br>";

   echo "<b><u> Assignment Operators </u></b><br/>";
   echo "Add = ".($x+=$y)." <br>";
   echo "Sub = ".($x-=$y)." <br>";
   echo "Mul = ".($x*=$y)." <br>";
   echo "Div = ".($x/=$y)." <br>";

   echo "<b><u> Increment / Decrement Operators </u></b><br/>";
   echo "Pre-Increment = ".++$x." <br>";
   echo "Post - Increment = ".$x++." <br>";
   echo "After Increment = ".$x."<br/>";
```



```
   echo "<b><u> Relational Operators </u></b><br/>";
   echo " => $x<$y ".var_dump($x<$y)." <br>";
   echo " => $x>$y ".var_dump($x>$y)." <br>";
   echo " => $x<=$y ".var_dump($x<=$y)." <br>";
   echo " => $x>=$y ".var_dump($x>=$y)." <br>";
   echo " => $x!=$y ".var_dump($x!=$y)." <br>";

   echo "<b><u> Logical Operators </u></b><br/>";
   echo " => $x&&$y ".var_dump($x&&$y)." <br>";
   echo " =>  $x||$y  ".var_dump($x||$y)."  <br>";
   echo " => $x xor $y ".var_dump($x xor $y)." <br>";

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

Output:

```
Arithmetic Operators
Add = 10+5 = 15
```

Sub = 10-5 = 5
Mul = 10*5 = 50
Div = 10/5 = 2
Assignment Operators
Add = 15
Sub = 10
Mul = 50
Div = 10
Increment / Decrement Operators
Pre-Increment = 11
Post - Increment = 11
After Increment = 12
Relational Operators
bool(false) => 12<5
bool(true) => 12>5
bool(false) => 12<=5
bool(true) => 12>=5
bool(true) => 12!=5
Logical Operators
bool(true) => 12&&5
bool(true) => 12||5
bool(false) => 12 xor 5

4.1.7: PROGRAM CONTROL

There are two control flow statements are used in PHP:

1. Conditional or Decision making Statements
2. Looping Statements

1. CONDITIONAL STATEMENTS:

- ✓ Conditional statements are used to perform different actions based on different conditions.
- ✓ Very often when you write code, you want to perform different actions for different decisions. We can use conditional statements in our code to do this.

In PHP we have the following conditional statements:

- ➢ **if statement** - executes some code only if a specified condition is true
- ➢ **if...else statement** - executes some code if a condition is true and another code if the condition is false
- ➢ **if...elseif....else statement** - specifies a new condition to test, if the first condition is false
- ➢ **switch statement** - selects one of many blocks of code to be executed

1. if Statement

The if statement is used to execute some code only if a specified condition is true.

Syntax:

```
if (condition) {
    code to be executed if condition is true;
}
```

Example:

```
<?php
$t = date("H");

if ($t < "12") {
    echo "Good Morning!";
}
?>
```

Output:

Good Morning!

2. if...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is false.

Syntax


```
if (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}
```

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```
<?php
$t = date("H");

if ($t <  "20") {
     echo "Have a good day!";
} else {
     echo "Have a good night!";
}
?>
```

Output:

Have a good day!

3. if...elseif....else Statement

Use the if....elseif...else statement to specify a new condition to test, if the first condition is false.

Syntax

```
if (condition) {
    code to be executed if condition is true;
} elseif (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!"

```
<?php
$t = date("H");
echo "<p>The hour (of the server) is " . $t;
echo ", and will give the following message:</p>";
```



```
if ($t < "10") {
    echo "Have a good morning!";
} elseif ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

Output:

The hour (of the server) is 03, and will give the following message:

Have a good morning!

4. switch Statement

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

Syntax

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
```

```
        case label3:
            code to be executed if n=label3;
            break;
        ...
        default:
            code to be executed if n is different from all labels;
    }
```

How it works?

- ✓ First we have a single expression n (most often a variable), that is evaluated once.
- ✓ The value of the expression is then compared with the values for each case in the structure.
- ✓ If there is a match, the block of code associated with that case is executed.
- ✓ Use break to prevent the code from running into the next case automatically.
- ✓ The default statement is used if no match is found.

Example:

```
<?php
$favcolor = "red";

switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}
?>
```


Output:

Your favorite color is red!

2. LOOPING STATEMENTS:

- ✓ Looping statements are used to perform certain actions repeatedly when the specified condition is true.
- ✓ **Loops:**

  Often when we write code, we want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

- ✓ In PHP, we have the following looping statements:
  - ❖ **while** - loops through a block of code as long as the specified condition is true
  - ❖ **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
  - ❖ **for** - loops through a block of code a specified number of times
  - ❖ f**oreach** - loops through a block of code for each element in an array

1. while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
    code to be executed;
}
```

The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

```
<?php
$x = 1;

while($x <= 5) {
    echo "The number is: $x <br>";
    $x++;
}
?>
```


Output:

The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5

2. do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do {
    code to be executed;
} while (condition is true);
```

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

```
<?php
$x = 1;
```

```
do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?>
```

Output:

The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5

3. for Loop

- ✓ PHP for loops execute a block of code a specified number of times.
- ✓ The for loop is used when you know in advance how many times the script should run.
- ✓ Syntax

```
for (init counter; test counter; increment counter) {
code to be executed;
}
```

- ✓ Parameters:
  - ***init counter:*** Initialize the loop counter value
  - ***test counter:*** Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
  - ***increment counter:*** Increases the loop counter value

The example below displays the numbers from 0 to 10:

```
<?php
for ($x = 0; $x <= 5; $x++) {
    echo "The number is: $x <br>";
}
?>
```

Output:

The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5

4. foreach Loop:

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value) {
    code to be executed;
}
```

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array ($colors):

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
    echo "$value <br>";
}
?>
```

Output:

red
green
blue
yellow

4.1.8: ARRAYS

An array is a data structure that stores one or more similar type of values in a single value

There are three different kinds of arrays and each array value is accessed using an ID which is called array index.

1) **Numeric array** - An array with a numeric index. Values are stored and accessed in linear fashion.
2) **Associative array** - An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
3) **Multidimensional array** - An array containing one or more arrays and values are accessed using multiple indices

CREATING ARRAYS:

In PHP, the array() function is used to create an array: array();

Syntax: **$array_variable = array("value1", "value2", ..., "valueN");**

1. Numeric Array

- ✓ These arrays can store numbers, strings and any object but their index will be represented by numbers.
- ✓ By default array index starts from zero.
- ✓ Example

Following is the example showing how to create and access numeric arrays.

```
<html>
<body>
<?php
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
$i=0;
foreach( $numbers as $value )
{
  echo "numbers[$i] = $value <br />";
```

```
  $i++;
}
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";

$i=0;
foreach( $numbers as $value )
{
 echo "numbers[$i] = $value <br />";
 $i++;
}
?>
</body>
</html>
```

Output:

```
numbers[0] = 1
numbers[1] = 2
numbers[2] = 3
numbers[3] = 4
numbers[4] = 5
numbers[0] = one
numbers[1] = two
numbers[2] = three
numbers[3] = four
numbers[4] = five
```

2. Associative Arrays

- ✓ The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index.
- ✓ Associative array will have their index as string so that you can establish a strong association between key and values.
- ✓ To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.
- ✓ Example:

```
<html>
<body>
<?php
/* First method to associate create array. */
$salaries = array(
                "AAA" => 2000,
                "BBB" => 1000,
                "CCC" => 500
               );

echo "Salary of AAA is ". $salaries['AAA'] . "<br />";
echo "Salary of BBB is ". $salaries['BBB']. "<br />";
echo "Salary of CCC is ". $salaries['CCC']. "<br />";
```

```
/* Second method to create array. */
$salaries['AAA'] = "high";
$salaries['BBB'] = "medium";
$salaries['CCC'] = "low";

foreach($salaries as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
</body>
</html>
```

Output:

Salary of AAA is 2000
Salary of BBB is 1000
Salary of CCC is 500
Key=AAA, Value=high
Key=BBB, Value=medium
Key=CCC, Value=low

3. Multidimensional Arrays

- ✓ A multi-dimensional array each element in the main array can also be an array.
- ✓ Each element in the sub-array can be an array, and so on.
- ✓ Values in the multi-dimensional array are accessed using multiple index.


```
<html>
  <body>

    <?php
      $marks = array(
        "AAA" => array
        ( "physics" => 35,
          "maths" => 30,
          "chemistry" => 39 ),

        "BBB" => array
        ( "physics" => 30,
          "maths" => 32,
          "chemistry" => 29 ),

        "CCC" => array
        ( "physics" => 31,
          "maths" => 22,
          "chemistry" => 39 )
      );
```

```
        /* Accessing multi-dimensional array values */
        echo "Marks for AAA in physics : " ;
        echo $marks['AAA']['physics'] . "<br />";

        echo "Marks for BBB in maths : ";
        echo $marks['BBB']['maths'] . "<br />";

        echo "Marks for CCC in chemistry : " ;
        echo $marks['CCC']['chemistry'] . "<br />";
      ?>

    </body>
</html>
```

Output:

Marks for AAA in physics : 35
Marks for BBB in maths : 32
Marks for CCC in chemistry : 39

4.1.9: FUNCTIONS

Definition: A function is a piece of code which takes one or more input in the form of parameter and does some processing and returns a value.

Creating functions:

Syntax:

```
function functionName(parameter_list)
{
    Function body
}
```

Example:

```
<html>

  <head>
    <title>Writing PHP Function with Parameters</title>
  </head>

  <body>
```

```
<?php
  function addFunction($num1, $num2)      // function definition
  {
    $sum = $num1 + $num2;
    echo "Sum of the two numbers is : $sum";
  }

  addFunction(10, 20); // function call
?>

  </body>
</html>
```

Output:

Sum of the two numbers is : 30

Passing Arguments by Reference:

- ✓ It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.
- ✓ Any changes made to an argument in these cases will change the value of the original variable.
- ✓ You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.
- ✓ Example:

```
<html>
    <head>
        <title>Passing Argument by Reference</title>
    </head>
    <body>
    <?php
        function addFive($num)
        {   $num += 5;  }

        function addSix(&$num)
        {
            $num += 6;
        }
        $orignum = 10;
        addFive( $orignum );
```

```
                    echo "Original Value is $orignum<br />";

                    addSix( $orignum );
                    echo "Original Value is $orignum<br />";
                    ?>
                </body>
            </html>
```

This will display following result –

Original Value is 10
Original Value is 16

PHP Functions returning value:

- ✓ A function can return a value using the return statement in conjunction with a value or object.
- ✓ return stops the execution of the function and sends the value back to the calling code.
- ✓ You can return more than one value from a function using return array(1,2,3,4).
- ✓ Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that return keyword is used to return a value from a function.

```
<html>
  <head>
        <title>Writing PHP Function which returns value</title>
  </head>
  <body>

   <?php
        function addFunction($num1, $num2)
       {
          $sum = $num1 + $num2;
        return $sum;
        }
        $return_value = addFunction(10, 20);

        echo "Returned value from the function : $return_value";
   ?>

   </body>
</html>
```

This will display following result –
Returned value from the function : 30

Setting Default Values for Function Parameters

We can set a parameter to have a default value if the function's caller doesn't pass it.

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

<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // this function call will use the default value „50"
setHeight(135);
setHeight(80);
?>
```



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

Output:

The height is : 350
The height is : 50
The height is : 135
The height is : 80

4.1.10: PHP BUILT-IN FUNCTIONS

- Array Functions
- Class/Object Functions
- Character Functions
- Date & Time Functions
- Error Handling Functions
- MySQL Functions
- ODBC Functions
- String Functions
- XML Parsing Functions

1. Array Functions:

These functions allow you to interact with and manipulate arrays in various ways. Arrays are essential for storing, managing, and operating on sets of variables.

Function	Description
array()	Create an array
array_chunk()	Splits an array into chunks of arrays
array_combine()	Creates an array by using one array for keys and another for its values
array_count_values()	Returns an array with the number of occurrences for each value
array_key_exists()	Checks if the specified key exists in the array
array_keys()	Returns all the keys of an array
array_merge()	Merges one or more arrays into one array
array_pop()	Deletes the last element of an array
array_product()	Calculates the product of the values in an array
array_push()	Inserts one or more elements to the end of an array
array_rand()	Returns one or more random keys from an array
array_reverse()	Returns an array in the reverse order
array_search()	Searches an array for a given value and returns the key
array_shift()	Removes the first element from an array, and returns the value of the removed element
array_slice()	Returns selected parts of an array
array_splice()	Removes and replaces specified elements of an array
array_sum()	Returns the sum of the values in an array
sort()	Sorts an array

```
<?php
   $a = array('green', 'red', 'yellow');
   $b = array('avocado', 'apple', 'banana');
   $c = array_combine($a, $b);
   echo "Comibination of Two Arrays : ";
   print_r($c);

   $c = array('avocado','watermelon','banana');
   echo "<br>";
   echo "Difference of Two Arrays : ";
   print_r(array_diff($b,$c));

   $d = array(10,4,2,7,1);
   echo "<br>Sum of array = ";
   print_r(array_sum($d));
```

```
echo "<br>product of array = ";
print_r(array_product($d));

echo "<br>Sorting of array = ";
sort($d);
print_r($d);
?>
```

Output:

```
Comibination of Two Arrays :
Array ( [green] => avocado [red] => apple [yellow] => banana )
Difference of Two Arrays : Array ( [1] => apple )
Sum of array = 24
product of array = 560
Sorting of array = Array ( [0] => 1 [1] => 2 [2] => 4 [3] => 7 [4] => 10 )
```

2. Class/Object Functions

- ✓ These functions allow you to obtain information about classes and instance objects.
- ✓ You can obtain the name of the class to which an object belongs, as well as its member properties and methods.

Function	Description
class_exists()	Checks if the class has been defined
get_class_methods()	Gets the class methods' names
get_class_vars()	Get the default properties of the class
get_class()	Returns the name of the class of an object
get_object_vars()	Gets the properties of the given object
get_parent_class()	Retrieves the parent class name for object or class
is_a()	Checks if the object is of this class or has this class as one of its parents
method_exists()	Checks if the class method exists
property_exists()	Checks if the object or class has a property

Example:

```
<?php
  class Box  {
     var $l = 10;
     var $b = 20;
    function volume()
     {
        $vol=$l*$b;
```

```
    }
  }

  $obj = new Box();
  $class_vars = get_class_vars(get_class($obj));
  $class_methods = get_class_methods(new Box());

  echo "Variables in the class <br>";
  foreach ($class_vars as $name => $value) {
   echo "$name : $value <br>";
  }

   echo "Methods in the class <br>";
  foreach ($class_methods as $method_name) {
   echo "$method_name <br>";
  }
?>
```

Output:

```
Variables in the class
l : 10
b : 20
Methods in the class
volume
```


3. Date Functions

These functions allow you to get the date and time from the server where your PHP scripts are running.

Function	Description
date_modify()	Alters the timestamp
date_sunrise()	Returns the time of sunrise for a given day / location
date_sunset()	Returns the time of sunset for a given day / location
date_timezone_set()	Sets the time zone for the DateTime object
date()	Formats a local time/date
getdate()	Returns an array that contains date and time information for a Unix timestamp
gettimeofday()	Returns an array that contains current time information
gmdate()	Formats a GMT/UTC date/time
localtime()	Returns an array that contains the time components of a Unix timestamp
time()	Returns the current time as a Unix timestamp

Example:

```
<?php
 print date('r');
 print "<br>";
 print date('D, d M Y H:i:s T');
 print "<br>";
?>
```

Output:

```
Thu, 03 Sep 2015 10:42:58 +0000
Thu, 03 Sep 2015 10:42:58 UTC
```

4. String Functions

String functions are used to manipulate strings.

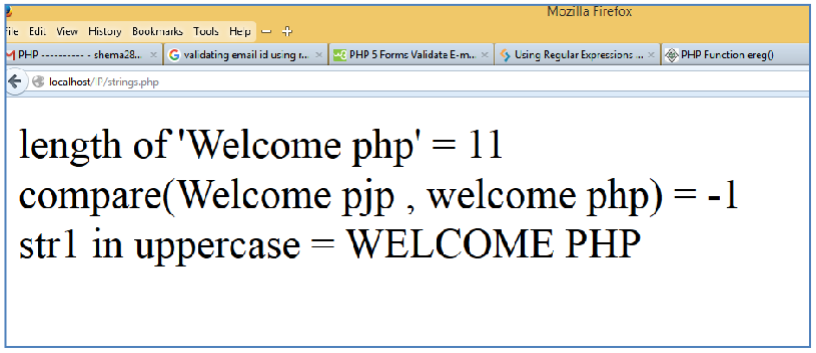
Function	Description
chr	returns a specific character
count_chars	return information about characters used in string
echo	output one or more strings
fprintf	write a formatted string to a stream
str_pad	pad a string to a certain length with another string
str_repeat	repeat a string
str_replace	replace all occurrences of the search string with the replacement string
str_ireplace	case-insensitive version str_replace
strlen	get string length
strcmp	string comparison
strncmp	string comparison of the first n characters
strpos	find the position of the first occurrence of a substring in a string

Example:

```
<html>
<body>
<?php
  $str1="Welcome php";
  echo "length of 'Welcome php' = ".strlen($str1)."<br>";

  $str2="welcome php";
  echo "compare(Welcome pjp , welcome php) =
".strcmp($str1,$str2)."<br>";

  echo "str1 in uppercase = ".strtoupper($str1)."<br>";
 ?>
```

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

Output:



4.1.12: PHP FORM HANDLING

Form: A Document that containing black fields, that the user can fill the data or user can select the data.

What is Validation?

Validation means check the input submitted by the user.

There are two types of validation are available in PHP. They are as follows –

- **Client-Side Validation** – Validation is performed on the client machine web browsers.
- **Server Side Validation** – After submitted by data, The data has sent to a server and perform validation checks in server machine.

Some of Validation rules for field

Field	Validation Rules
Name	Should require letters and white-spaces
Email	Should require @ and .
Website	Should require a valid URL
Radio	Must be selectable at least once
Check Box	Must be checkable at least once
Drop Down menu	Must be selectable at least once

Collecting form data in PHP:

The PHP superglobals $_GET and $_POST are used to collect form-data.

superglobal arrays: superglobal arrays are used to obtain client information from the request. They are associative arrays that hold variables acquired from user input.

Other useful superglobal arrays:

variable name	Description
$_SERVER	Data about the currently running server
$_ENV	Data about the client‟s environment
$_GET	Data sent to the server by a get request
$_POST	Data sent to the server by a post request
$_COOKIE	Data contained in the cookies on the client computer
$_GLOBALS	Array containing all global variables

When to use GET?

- ✓ Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL).
- ✓ GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
- ✓ GET may be used for sending non-sensitive data.

When to use POST?

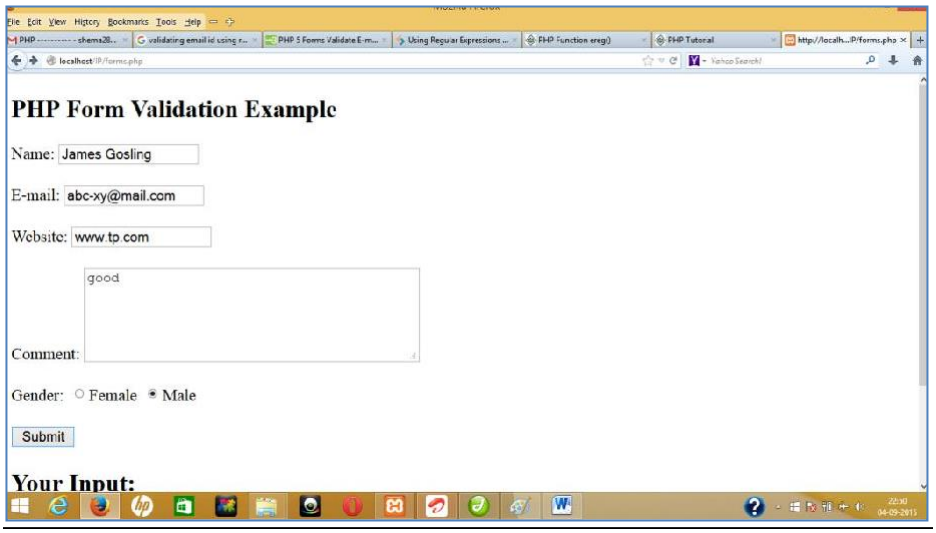
- ✓ Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.

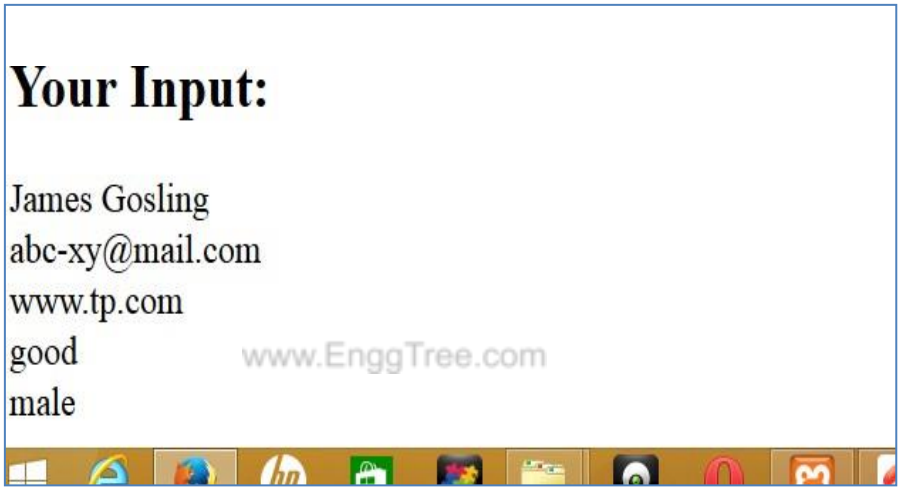
Example:

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>

<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $name = $_POST["name"];
   $email = test_input($_POST["email"]);
   $website = $_POST["website"];
   $comment = $_POST["comment"];
   $gender = $_POST["gender"];
}
```

```
function  test_input($data)  {
  if(ereg("^[a-zA-Z0-9._-]+@[a-zA-Z0-9]+.[a-z.]{2,5}$",$data))
   return "<script>alert('valid Email  Id')</script>".$data;
   else
   return "false";
  }
?>

<h2>PHP Form Validation Example</h2>
<form method="post" action="forms.php">
   Name: <input type="text" name="name">
   <br><br>
   E-mail: <input type="text" name="email">
   <br><br>
   Website: <input type="text" name="website">
   <br><br>
   Comment: <textarea name="comment" rows="5" cols="40"></textarea>
   <br><br>
   Gender:
   <input type="radio" name="gender" value="female">Female
   <input type="radio" name="gender" value="male">Male
   <br><br>
   <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

Output:





4.1.15: Connecting Database

3. How will you Connect a PHP program with database? Explain with example application.

Answer:

PHP: Hypertext Preprocessor.

PHP is a widely-used, open source, Server-side Scripting language for creating dynamic web pages. PHP Scripts are executed on the Server.

MYSQL: MySQL is the most popular open source Relational SQL database management system. MySQL is one of the best RDBMS being used for developing web-based Software applications.

Steps to Connect database with PHP program:

1. Make the connection
2. Select the database to use.
3. Perform database query.
4. Put the data on the page.
5. close the connection
6. Styling the data.

1) Make the Connection:

```
$db = mysql_connect ("localhost", "username",
                     "password");
```

2) Selecting the database to use:

```
$db1= mysql_select_db ("database_name", $db);
```

3) Perform database Query: (10)

```
$result = mysql_query ("SELECT * FROM mytable",
                       $db);
```

4) Put the data on the page:

```
while ($row = mysql_fetch_array ($result))
{
  echo $row[1]." ".$row[2]."<br/>";
}
```

5) Closing the connection:

```
mysql_close ($db);
```

6) Styling the data:

```
$result = mysql_query ("SELECT * FROM mytable", $db);
while ($row = mysql_fetch_array ($result))
{   echo "<h2>";
    echo $row[1]." ";
    echo "</h2>";
    echo "<p>";
    echo $row[2]." ";
    echo "</p>";
}
```

Example: PHP Script to create a new database and perform the following operation: (11)

1. Insert 2. Delete 3. Update 4. Display.

Php document [demo.php].

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "root", "mypassword") or
        die(mysql_error());
mysql_select_db("test") or die(mysql_error());
echo "Connected to database!";
mysql_query("CREATE TABLE mytable(id INT NOT NULL,
AUTO_INCREMENT, name VARCHAR(30), Phone INT, email VARCHAR(30))");
echo "Table Created"; print "<br/>";
// Insert rows into the table.
mysql_query("INSERT INTO mytable (name, phone, email)
      VALUES ('James', 11111, 'abc123@gmail.com')")
  or die(mysql_error());
mysql_query("INSERT INTO mytable (name, phone, email)
      VALUES ('Gosling', 22222, 'pqr234@gmail.com')")
  or die(mysql_error());
echo "Data Inserted"; print "<br/>";
$result = mysql_query("SELECT * FROM mytable");
      or die(mysql_error());
print "<br/>";
```

(12)

```
Print "<b> User Database</b>";
echo "<table border='1'>";
echo "<tr> <th> ID </th> <th> Name </th> <th> phone </th>
      <th> Email-ID </th> </tr>";
while ($row = mysql_fetch_array ($result))
{
   echo "<tr><td>". $row['id'] . "</td>"
   echo "<td>". $row['name'] . "</td>";
   echo "<td>". $row['phone'] . "</td>";
   echo "<td>". $row['email'] . "</td></tr>";
}
echo "</table>";
$result = mysql_query ("UPDATE mytable SET phone=55555
          WHERE phone=22222") or die(mysql_error());
Print "<br/>"; echo "Data Updated";
$result = mysql_query ("SELECT * FROM mytable");
      or die(mysql_error());
Print "<br/>";
Print "User Database after update";
echo "<table border='1'>";
while ($row = mysql_fetch_array ($result))
{   echo "<tr><td>". $row['id'] . "</td>";
    echo "<td>". $row['name'] . "</td>";
    echo "<td>". $row['phone'] . "</td>";
    echo "<td>". $row['mail'] . "</td></tr>";
}
echo "</table>";
```

(13)

```
$result = mysql_query("DELETE FROM mytable
        WHERE phone = 11111") or die(mysql_error());
print "<br/>";
echo " Data deleted";
$result = mysql_query("SELECT * FROM mytable")
        or die(mysql_error());
print "<br/>";
echo " User Database after delete";
echo "<table border='1'>";
while ($row = mysql_fetch_array($result))
{
    echo "<tr><td>" . $row['id'] . "</td>";
    echo "<td>" . $row['name'] . "</td>";
    echo "<td>" . $row['phone'] . "</td>";
    echo "<td>" . $row['email'] . "</td></tr>";
}
echo "</table>";
?>
```

output :

Mozilla Firefox

Connected to database!
Table Created
Data Inserted
User Database

Id	Name	Phone	Email-ID
1	James	11111	abc123@gmail.com
2	Gosling	22222	pqr834@gmail.com

Data Updated!
User Database after update

1	James	11111	abc123@gmail.com
2	Gosling	55555	pqr834@gmail.com

Data Deleted;
User Database after Delete

2	Gosling	55555	pqr234@gmail.com.

4.2.1: Basics of XML

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

Features of XML / Advantages / Uses

- ✓ Simplify the creation of HTML documents for large sites
- ✓ To exchange information between organizations
- ✓ Offload and reload databases
- ✓ Store and arrange data
- ✓ Any type of data can be expressed in XML
- ✓ Suits well for commerce applications, scientific purposes, mathematics, chemical formulae
- ✓ It can be used in handheld devices, smartphones, etc
- ✓ Hardware, software and language independent

4.2.2: XML Syntax Rules

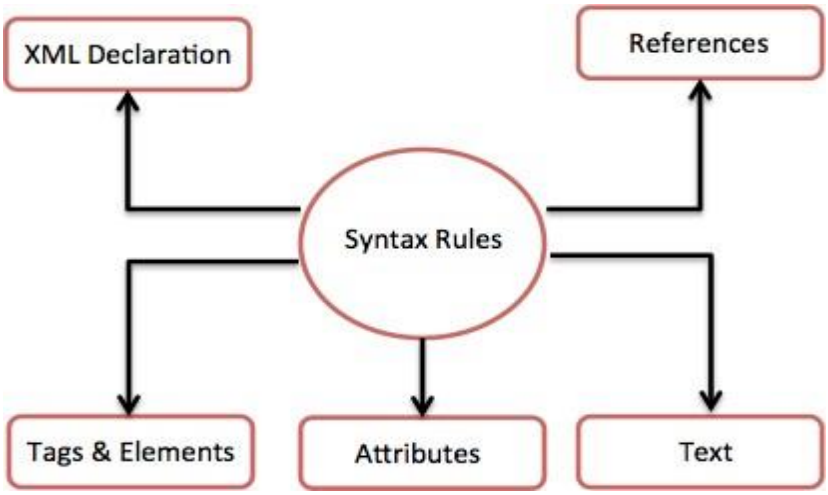
In this chapter, we will discuss the simple syntax rules to write an XML document. Following is a complete XML document –

```
<?xml version = "1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

You can notice there are two kinds of information in the above example –

- Markup, like <contact-info>
- The text, or the character data, Tutorials Point and (040) 123-4567.

The following diagram depicts the syntax rules to write different types of markup and text in an XML document.



Let us see each component of the above diagram in detail.

XML Declaration

The XML document can optionally have an XML declaration. It is written as follows –

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

Where *version* is the XML version and *encoding* specifies the character encoding used in the document.

Syntax Rules for XML Declaration

- The XML declaration is case sensitive and must begin with "**<?xml>**" where "**xml**" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of *encoding* that you put in the XML declaration.

Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. The names of XML-elements are enclosed in triangular brackets < > as shown below –

```
<element>
```

Syntax Rules for Tags and Elements

Element Syntax – Each XML-element needs to be closed either with start or with end elements as shown below –

```
<element> ….</element>
```

or in simple-cases, just this way –

```
<element/>
```

Nesting of Elements – An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

The Following example shows incorrect nested tags –

```
<?xml version = "1.0"?>
<contact-info>
<company>ABC
</contact-info>
</company>
```

The Following example shows correct nested tags –

```
<?xml version = "1.0"?>
<contact-info>
  <company>ABC</company>
<contact-info>
```

Root Element – An XML document can have only one root element. For example, following is not a correct XML document, because both the **x** and **y** elements occur at the top level without a root element –

```
<x>…</x>
<y>…</y>
```

The Following example shows a correctly formed XML document –

```
<root>
  <x>…</x>
  <y>…</y>
</root>
```

Case Sensitivity – The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case. For example, **<contact-info>** is different from **<Contact-Info>**

XML Attributes

An **attribute** specifies a single property for the element, using a name/value pair. An XML-element can have one or more attributes. For example –

abc Tutorial!

Here **href** is the attribute name and **http://www.abc.com/** is attribute value.

Syntax Rules for XML Attributes

- Attribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and *href* are considered two different XML attributes.
- Same attribute cannot have two values in a syntax. The following example shows incorrect syntax because the attribute *b* is specified twice –

```
<a b = "x" c = "y" b = "z"> .... </a>
```

- Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks. Following example demonstrates incorrect xml syntax –

```
<a b = x>.....</a>
```

In the above syntax, the attribute value is not defined in quotation marks.

XML References

References usually allow you to add or include additional text or markup in an XML document. References always begin with the symbol **"&"** which is a reserved character and end with the symbol **";".** XML has two types of references –

- **Entity References** – An entity reference contains a name between the start and the end delimiters. For example **&** where *amp* is *name*. The *name* refers to a predefined string of text and/or markup.
- **Character References** – These contain references, such as **A**, contains a hash mark ("#") followed by a number. The number always refers to the Unicode code of a character. In this case, 65 refers to alphabet "A".

XML Text

- ✓ The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case. To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files.
- ✓ Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored.
- ✓ Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly. To use them, some replacement-entities are used, which are listed below –

Not Allowed Character	Replacement Entity	Character Description
<	&lt;	less than
>	&gt;	greater than

&	&amp;	ampersand
'	&apos;	apostrophe
"	&quot;	quotation mark

XML Syntax Rules (Quick Recap..)

1) XML documents must have a root element.

XML documents must contain one root element that is the parent of all other elements:

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

2) The XML Prolog

This line is called the XML prolog:

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

The XML prolog is optional. If it exists, it must come first in the document.

3) All XML Elements Must Have a Closing Tag

In XML, it is illegal to omit the closing tag. All elements must have a closing tag:

** This is a paragraph. **
**
**

4) XML Tags are Case Sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>. Opening and closing tags must be written with the same case:

<message>This is correct</message>

"Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

5) XML Elements must be properly nested

In XML, all elements must be properly nested within each other:

The Following example shows correct nested tags –

```
<?xml version = "1.0"?>
<contact-info>
  <company>TutorialsPoint</company>
</contact-info>
```

6) XML Attribute values must always be quoted

XML elements can have attributes in name/value pairs just like in HTML.

In XML, the attribute values must always be quoted:

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

7) Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>salary < 1000</message>
```

To avoid this error, replace the "<" character with an entity reference:

```
<message>salary &lt; 1000</message>
```

8) Comments in XML

The syntax for writing comments in XML is similar to that of HTML:

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

Two dashes in the middle of a comment are not allowed:

```
<!-- This is an invalid -- comment -->
```

9) White-space is Preserved in XML

XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space):

XML:	Hello Tove
HTML:	Hello Tove

10) Well Formed XML

XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

❖ **Well Formed XML Vs Valid XML documents:**

Well Formed XML documents: XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

Valid XML Document:

If an XML document conforms to its DTD/Schema that defines the proper structure of the document, then the XML document is **valid XML document.**

An XML document can reference a **Document Type Definition (DTD)** or a **schema** that defines the proper structure of the XML document. When an XML document references a DTD or a schema, XML parsers (called **validating parsers)** can read the DTD/Schema and check that the XML document follows the structure defined by the DTD/Schema. If the XML document conforms to the DTD/schema, the XML document is valid.

DIFFERENCE BETWEEN XML AND HTML

XML	HTML
Software and hardware independent	Software and hardware dependent
To send and store data	To display data
Focus on what data is present	Focus on how data looks
It is a Framework for markup language definition	It is a markup language
Case sensitive	Case insensitive
Transport data between app and database	Design client side web programs
Custom tags allowed	Only predefined tags
Open and close tags are strict	Not strict
White space insensitive	White space insensitive
Carry information	Display information
Dynamic	Static

4.2.3: Document Type Definition (DTD)

- ✓ **DTD stands for Document Type Definition. An XML DTD defines the structure of an XML document.**
- ✓ **XML DTD is used to define the basic building block of any XML document. Using DTD, we can specify the various element types, attributes, and their relationships with one another.**
- ✓ **XML DTD is used to specify the set of rules for structuring data in any XML file.**
- ✓ **An XML document is considered "well formed" and "valid" if it is successfully validated against DTD.**

Basic Building Blocks of XML:

Various building blocks of XML are:

1. **Elements:-** Basic entity is element. Elements are used for defining the tags.

2. **Attribute:**- Used to specify the values of the element. It is to provide additional information to an element.
3. **CDATA:**- Character Data. CDATA will be parsed by the parser.
4. **PCDATA:**- Parsed character data (ie. Text).

An example of DTD

DTD is declared inside<!DOCTYPE> definition when the DTD declaration is internal. In this example we can see that there is an XML document that has a <!DOCTYPE> definition. The bold part in the following example is the DTD declaration.

```
<?xml version="1.0"?>
<!-- XML DTD declaration starts here -->
<!DOCTYPE beginnersbook [
<!ELEMENT beginnersbook (to,from,subject,message)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT message (#PCDATA)>
]>
<!-- XML DTD declaration ends here-->
<beginnersbook>
        <to>My Readers</to>
        <from>Chaitanya</from>
        <subject>A Message to my readers</subject>
        <message>Welcome to beginnersbook.com</message>
</beginnersbook>
```


Explanation:

- **!DOCTYPE beginnersbook** defines that this is the beginning of the DTD declaration and the root element of this XML document is beginnersbook
- **!ELEMENT beginnersbook** defines that the root element beginnersbook must contain four elements: "to,from,subject,message"
- **!ELEMENT to** defines that the element "to" is of type "#PCDATA" where "#PCDATA" stands for Parsed Character Data which means that this data is parsable by XML parser
- **!ELEMENT from** defines that the element "from" is of type "#PCDATA"
- **!ELEMENT subject** defines that the element "subject" is of type "#PCDATA"
- **!ELEMENT message** defines that the element "message" is of type "#PCDATA"

Types of DTD: 1. Internal DTD, 2. External DTD

1. **Internal DTD:** The DTD will be specified within the same XML document.
2. **External DTD:** The DTD is written in seperate file and saved with the extension ".dtd".
   The DTD file name must be specified in the corresponding XML file.

Example : Internal DTD:

Internal.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
<!ELEMENT student (name, rollno,
                   mark)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT rollno (#PCDATA)>
<!ELEMENT mark (#PCDATA)>
]>
<student>
  <name> XXXX </name>
  <rollno> 1001 </rollno>
  <mark> 79 </mark>
</student>
```

External DTD

Step 1: Creation of DTD file.

student.dtd

```
<!ELEMENT student (
      name, rollno, mark)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT rollno (#PCDATA)>
<!ELEMENT mark (#PCDATA)>
```

Step 2: Creation of XML document.

```
<?xml version="1.0" ?>
<!DOCTYPE student SYSTEM
        "student.dtd">
<student>
   <name> XXXX </name>
   <rollno> 1001 </rollno>
   <mark> 79 </mark>.
</student>
```

Advantages of DTD

- ✓ XML processor enforces structure, as defined in DTD
- ✓ Application is accessed easily in document structure
- ✓ DTD gives hint to XML processor
- ✓ Reduces size of document

4.2.4: XML Schema

XML schema is a language which is used for expressing constraint about XML documents. An XML schema is used to define the structure of an XML document. It is like DTD but provides more control on XML structure.

- ❖ XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data.
- ❖ The XML schema language is called as XML Schema Definition (XSD) Language.
- ❖ XML Schema defines elements, attributes, elements having child elements, order of child elements. It also defines fixed and default values of elements and attributes.
- ❖ XML schema also allows the developers to use data types.
- ❖ An XML document is called "well-formed" if it contains the correct syntax. A well-formed and valid XML document is one which has been validated against Schema.

XML Schema Example

Let's create a schema file.

employee.xsd

```
1. <?xml version="1.0"?>
2. <xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"    >
3. <xs:element name="employee">
4.   <xs:complexType>
5.     <xs:sequence>
6.       <xs:element name="firstname" type="xs:string"/>
7.       <xs:element name="lastname" type="xs:string"/>
8.       <xs:element name="email" type="xs:string"/>
9. </xs:sequence>
10.    </xs:complexType>
11.    </xs:element>
12.    </xs:schema>
```

Let's see the xml file using XML schema or XSD file.

employee.xml

```
<?xml version="1.0"?>
<employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="employee.xsd">

  <firstname>James</firstname>
  <lastname>Gosling</lastname>
  <email>James@abc.com</email>
</employee>
```

Description of XML Schema

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

<xs:element name="lastname" type="xs:string"/> : It defines that the element 'lastname' is of string/text type.
<xs:element name="email" type="xs:string"/> : It defines that the element 'email' is of string/text type.

XML Schema Definition Types

You can define XML schema elements in following ways:

1. **Simple Type** –
   - ✓ A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.
   - ✓ It contains less attributes, child elements and cannot be left empty.
   - ✓ Simple type element is used only in the context of the text.
   - ✓ The syntax for defining a simple element is:

     <xs:element name="xxx" type="yyy"/>
   - ✓ Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date.
   - ✓ For example:

     <xs:element name="phone_number" type="xs:int" />

2. **Complex Type** –
   - ✓ A complex type is a container for other element definitions.
   - ✓ This allows you to specify which child elements an element can contain and to provide some structure within your XML documents.
   - ✓ For example:

```
<xs:element name="Address">
<xs:complexType>
    <xs:sequence>
        <xs:element name="name" type="xs:string" />
        <xs:element name="company" type="xs:string" />
        <xs:element name="phone" type="xs:int" />
    </xs:sequence>
</xs:complexType>
</xs:element>
```

3. Global Types –

- ✓ With global type, you can define a single type in your document, which can be used by all other references.
- ✓ For example, suppose you want to generalize the person and company for different addresses of the company. In such case, you can define a general type as below:

```
<xs:element name="AddressType">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```


Now let us use this type in our example as below:

```
<xs:element name="Address1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="address" type="AddressType" />
            <xs:element name="phone1" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Address2">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="address" type="AddressType" />
            <xs:element name="phone2" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- Instead of having to define the name and the company twice (once for Address1 and once for Address2), we now have a single definition.
- This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Data Types in XML Schema

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

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

i) <xs:string> data type

- ✓ The <xs:string> data type can take characters, line feeds, carriage returns, and tab characters.
- ✓ The XML processor does not replace line feeds, carriage returns, and tab characters in the content with space and keep them intact.
- ✓ For example, multiple spaces or tabs are preserved during display.

ii) <xs:date> data type

- ✓ The <xs:date> data type is used to represent date in YYYY-MM-DD format.
  - YYYY – represents year
  - MM – represents month
  - DD – represents day

iii) <xs:numeric> data type

- ✓ The <xs:decimal> data type is used to represent numeric values.
- ✓ It supports decimal numbers up to 18 digits.
- ✓ The <xs:integer> data type is used to represent integer values.

iv) <xs:boolean> data type

- ✓ The <xs:boolean> data type is used to represent true, false, 1 (for true) or 0 (for false) value.

Example

Here are some XML elements:

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

And here are the corresponding simple element definitions:

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

Default and Fixed Values for Simple Elements

- ✓ Simple elements may have a default value OR a fixed value specified.
- ✓ A default value is automatically assigned to the element when no other value is specified.
- ✓ In the following example the default value is "red":

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

- ✓ A fixed value is also automatically assigned to the element, and you cannot specify another value.
- ✓ In the following example the fixed value is "red":

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

Example: XML document using XML Schema

studentSchema.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:="http://www.w3.org/2001/XMLSchema">
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="Rollno" type="xs:integer" />
        <xs:element name="Mark" type="xs:integer" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

MyScheme.xml [xml document that uses the qStudentSchema.xsd].

```
<?xml version="1.0" encoding="UTF-8"?>
<Student xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchema
Location="qStudentSchema.xsd">
<Name> ABC </Name>
<Rollno> 1001 </Rollno>
<Mark> 80 </Mark>
</Student>.
```

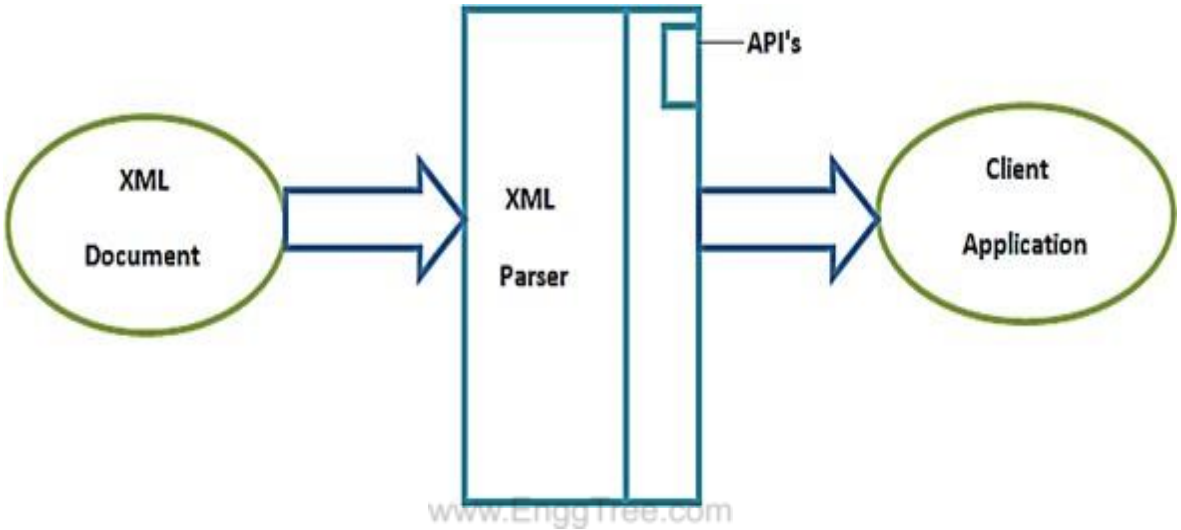
DIFFERENCE BETWEEN XML DTD & XML SCHEMA DEFINITION (XSD)

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

4.2.6: XML Parsers and Validation

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document (Well Formed Document) and may also validate the XML documents (Valid XML Document).

XML parser validates the document and check that the document is well formatted.



Types of XML Parsers

The primary goal of any XML processor is to parse the given XML document. Java has a rich source of built-in APIs for parsing the given XML document. These are the two main types of XML Parsers:

1. DOM Parser (Tree based)
2. SAX Parser (Event Based)

S.No.	DOM API	SAX API
1.	Document Object Model	Simple API for XML
2.	Tree Based Parsing	Event Based Parsing
3.	The entire XML document is stored in memory before actual processing. Hence it requires more memory	Part of XML document is stored in memory. Because the parsing is done by generating the sequence of events or by calling handler functions. Hence it requires less memory

4.	The DOM approach is useful for smaller applications because it is simpler to use but it is certainly not used for larger XML documents because it will then require larger amount of memory	The SAX approach is useful for parsing the large XML document because it is event based, XML gets parsed node by node and does not require large amount of memory
5.	We can insert or delete a node	We can insert or delete a node
6.	Traversing is done in any direction in DOM approach	Top to bottom traversing is done in SAX approach

1. DOM (Document Object Model) Parser

- ✓ A DOM document is an object which contains all the information of an XML document. It is composed like a tree structure. The DOM Parser implements a DOM API. This API is very simple to use.
- ✓ The DOM API is defined in the package **or.w3c.dom.***
- ✓ Using this API a DOM object tree is created from the input XML document and this tree helps in parsing the XML document.

Example: Checking the Well Formedness of XML document using DOM API.

dom.java

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;

public class dom
{
public static void main(String bala[])
{
try
{
System.out.println("Enter XML document name");
BufferedReader input = new BufferedReader( new InputStreamReader(System.in));
String filename = input.readLine();
File fp = new File(filename);
if(fp.exists())
{
try
{
DocumentBuilderFactory dbf = new DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.new DocumentBuilder();
InputSource ips = new InputSource(filename);
```

```
Document doc = db.parse(ips);
System.out.println(filename + “is well formed”);
}
catch(Exception  e)
{
 System.out.println(filename+“ isn‟t well-formed”);
 System.exit(1);
}
}
else
{
System.out.println(“File  not  Found”);
}
}
catch(IOException ioe)
{
ioe.printStackTrace();
}
}
}
```

User.xml


```
<?xml version="1.0"?>
<userdata>
        <user1>
           <userno>001</userno>
           <username>Bala</username>
           <phonenumner>123456789</phonenumber>
           <address>Chennai</Chennai>
        </user1>
        <user2>
           <userno>002</userno>
           <username>Suresh</username>
           <phonenumner>987654321</phonenumber>
           <address>madurai</Chennai>
        </user2>
        <user3>
           <userno>003</userno>
           <username>arul</username>
           <phonenumner>1122334455</phonenumber>
           <address>Vellore</Chennai>
        </user3>
</userdata>
```

Output:

C:> javac dom.java
C:> java dom
Enter file name dom.xml
dom.xml is well formed

Program Explanation:

Program Explanation

1. In above Java program we have to import some useful packages initially. Those described as follows -

- The **java.io** package provides the interface for performing simple input and output operations.
- The **javax.xml.parsers.*** package provides the classes allowing the processing of XML documents. It supports various classes such as **DocumentBuilder** and **DocumentBuilderFactory**.
- The pacakge **org.w3c.dom.*** provides the interfaces for Document Object Model which is a component API of JAVA API for XML processing.
- The package **org.xml.sax.*** provides the classes and interfaces for the Simple API for XML (SAX) which is a component API of JAVA API.

2. Reading the name of XML document using the command prompt. Using the input stream for the **BufferReader** class we can read the contents from the command prompt.
3. **DocumentBuilderFactory** is a factory API an application can obtain parser. This parser basically produces DOM object tree from given XML document. Using the object of DocumentBuilderFactory an instance for **DocumentBuilder** is created. The object of **DocumentBuilder** is used to invoke a method **parse**. This method takes as XML document as an input, parses it. If the XML document is well formed then appropriate message will be displayed. In an XML document if every starting tag has an ending tag then that document is said to be well formed otherwise it is not.
4. In the **try** block we are calling the method **parse** for parsing the XML document. If the XML document is not well formed then the control of the program will go to **catch** block.

We will consider following XML document for executing above program -

student1.xml

```
<?xml version="1.0" ?>
<student>
   <Roll_No>10</Roll_No>
   <Personal_Info>
      <Name>Parth</Name>
      <Address>Pune</Address>
      <Phone>1234567890</Phone>
   </Personal_Info>
   <class>Second</class>
   <Subject>Mathematics</Subject>
   <Marks>100
   <Roll_No>20</Roll_No>
   <Personal_Info>
      <Name>Anuradha</Name>
      <Address>Banglore</Address>
      <Phone>90901233</Phone>
   </Personal_Info>
   <class>Fifth</class>
   <Subject>English</Subject>
   <Marks>90</Marks>

   <Roll_No>30</Roll_No>
   <Personal_Info>
      <Name>Anand</Name>
      <Address>Mumbai</Address>
      <Phone>90901256</Phone>
   </Personal_Info>
   <class>Fifth</class>
   <Subject>English</Subject>
   <Marks>90</Marks>
</student>
```

Purposely made this statement like this! (It is not well formed)

Output:

C:> java dom
Enter file name student1.xml
[FatalError] student1.xml:34:3: The element type "Marks" must be terminated by the matching end-tag "</Marks>".
Student1.xml isn‟t well-formed!

Advantages

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

Disadvantages

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

2. SAX (Simple API for XML) Parser

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

Example: Checking the Well Formedness of XML document using SAX API.

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class Parsing_SAXDemo
{
public static void main(String bala[])
{
try
{
System.out.println("Enter XML document name");
BufferedReader input = new BufferedReader( new InputStreamReader(System.in));
String filename = input.readLine();
File fp = new File(filename);
```

```
if(fp.exists())
{
try
{
XMLReader reader = XMLReaderFactory.CreateXMLReader();
reader.parse(filename);
System.out.println("filename + "is well formed");
}
catch(Exception  e)
{
System.out.println("filename + "is not well formed");
System.exit(1);
}
}
else
{
System.out.println("file not found");
}
}

catch(IOException ioe)
{
ioe.printStackTrace();
}
}
}
```


Output:

student.xml

```
<?xml version="1.0" ?>
<student>
   <Roll_No>10</Roll_No>
   <Personal_Info>
      <Name>Parth</Name>
      <Address>Pune</Address>
      <Phone>1234567890</Phone>
   </Personal_Info>
   <class>Second</class>
   <Subject>Mathematics</Subject>
   <Marks>100</Marks>
</student>
```

o/p:

```
$ java Parsing_SAXDemo
Enter the name of XML
document : student.xml
student.xml is well-formed.
```

Program Explanation

1. In above Java program we have to import some useful packages initially. Those are described as follows -
   - The **java.io** package provides the interface for performing simple input and output operations.
   - The package **org.xml.sax.*** provides the classes and interfaces for the Simple API for XML (SAX) which is a component API of JAVA API.
   - The package **org.xml.sax.helpers.*** provides the helper classes for Simple API for XML(i.e. SAX) which is a component of JAVA API for processing XML.
2. Reading the name of XML document using the command prompt. Using the input stream for the **BufferReader** class we can read the contents from the command prompt.
3. The **XMLReaderFactory** helps in creating an XML reader. This reader then parses xml document using the appropriate callbacks. Using **createXMLReader** an object **reader** is created using which we can call the method **parse** for parsing the XML document.
4. In the **try** block we are calling the method **parser** for parsing the XML document. If the XML document is not well formed then the control of the program will go to **catch** block.

Advantages

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

Disadvantages

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

4.2.7: XSL and XSLT Transformation

EXtensible Stylesheet Language (XSL) and XSL Transformations(XSLT)

EXtensible Stylesheet Language (XSL) documents describes how an XML document should look on the browser.

XSL is a group of three technologies:

1. XSL-FO(XSL Formatting Objects) – It is vocabulary for specifying formatting of an XML document.

2. XPath (XML Path Language) – It is string-based language of expressions for effectively locating specific elements and attributes in XML documents. It is used for navigating through XML document.

3. XSLT (XSL Transformations) – It is a technology for transforming XML documents into other documents; i.e., transforming the structure of the XML document data to another structure.

- ✓ Transforming an XML document using XSLT involves two tree structures:
  [1] **Source Tree –** the XML document to be transformed.
  [2] **Result Tree –** the XML document to be created.

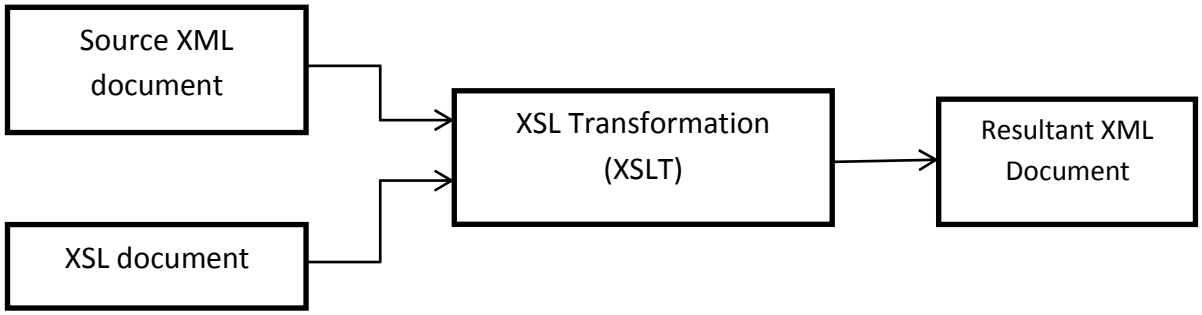


Fig: Processing of XML document using XSLT

- ✓ **XPath** is used to locate parts of the source-tree document that match **templates** defined in an **XSL stylesheet.**
- ✓ When a match occurs (i.e., a node matches a template), the matching template executes and adds its result to the **result tree.**
- ✓ When there are no more matches, XSLT has transformed the source tree into the result tree.
- ✓ XSLT does not analyse every node of the source tree; it selectively navigates the source tree using XPath‟s **select** and **match** attributes.
- ✓ For XSLT to work, the source tree must be properly structured. Schemas, DTDs and validating parsers can validate the document structure before using XPath and XSLT.

XSL Elements:

Elements	Description
<xsl:template>	Contains rules to apply when a specified node is matched. The "**match**" attribute is associated with this element. The **match= "/"** defines the whole document.
<xsl:value-of select = "expression">	Selects the value of an XML element and adds it to the output tree. The **select** attribute contains an XPath expression.
<xsl:for-each select = "expression">	Applies a template to every node selected by the XPath expression.
<xsl:sort select = "expression">	Sorts the nodes selected by <<xsl:for-

	each select = “expression”> element.
<xsl:output>	Used to define the format of the output document.
<xsl:copy>	Adds the current node to the output tree.

✓ **Example: XSL Transformation**
Student.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="stud.xsl" ?>
<student-details>
    <student>
        <name>AAA</name>
        <reg_no>1001</reg_no>
        <year>1</year>
        <semester>II</semester>
        <grade>B</grade>
        <address>chennai</address>
    </student>
    <student>
        <name>BBB</name>
        <reg_no>1002</reg_no>
        <year>2</year>
        <semester>III</semester>
        <grade>B</grade>
        <address>chennai</address>
    </student>
    <student>
        <name>CCC</name>
        <reg_no>1003</reg_no>
        <year>3</year>
        <semester>V</semester>
        <grade>A</grade>
        <address>chennai</address>
    </student>
    <student>
        <name>DDD</name>
        <reg_no>2001</reg_no>
        <year>1</year>
        <semester>II</semester>
        <grade>A</grade>
```

```
        <address>chennai</address>
    </student>
    <student>
        <name>EEE</name>
        <reg_no>2002</reg_no>
        <year>1</year>
        <semester>II</semester>
        <grade>C</grade>
        <address>chennai</address>
    </student>

</student-details>
```

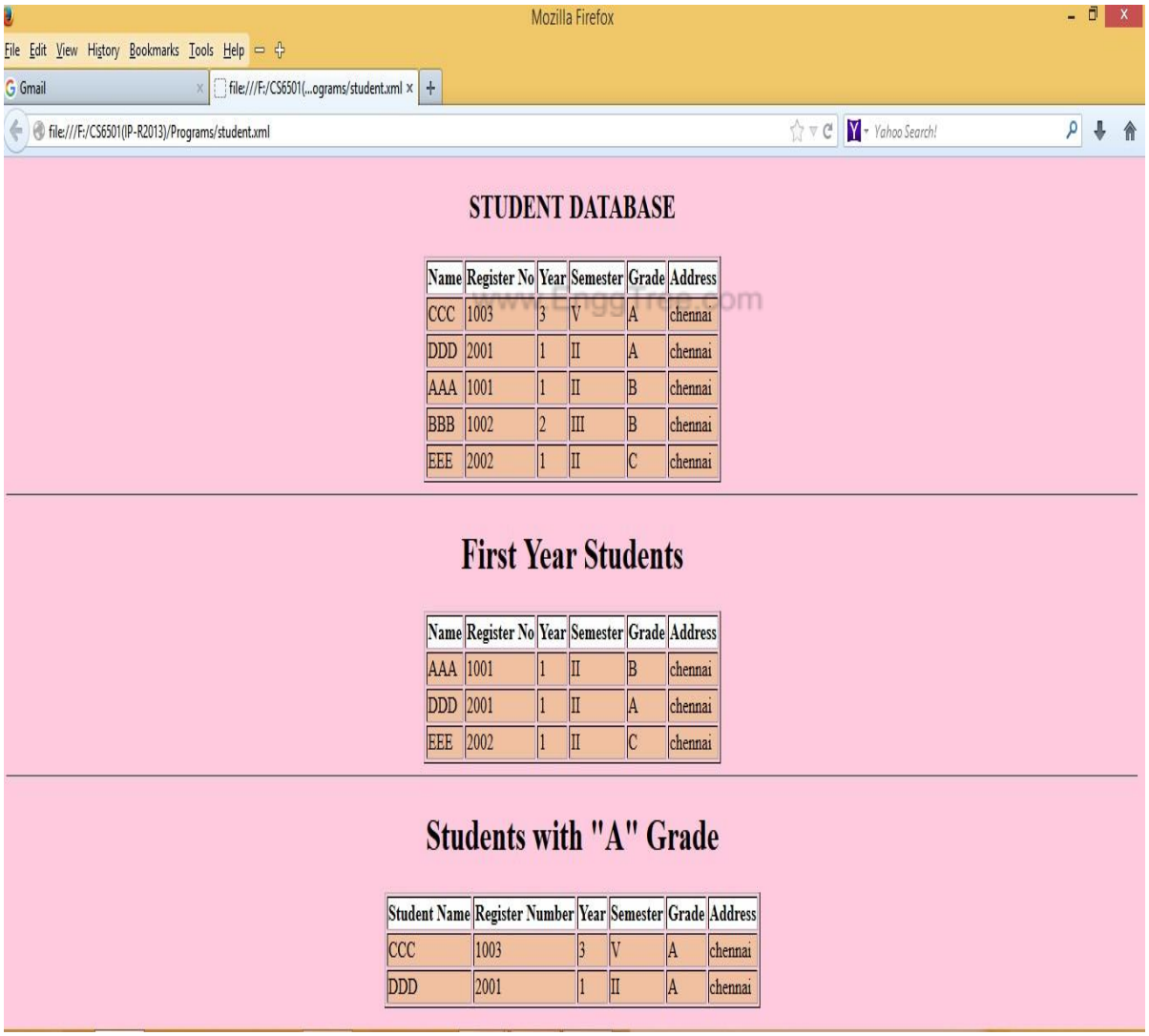
Stud.xsl

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
 <body bgcolor="#ffcadd">
  <center>
  <h2> STUDENT DATABASE </h2>
  <table border="1">
   <tr bgcolor="white">
    <th>Name</th>
    <th>Register No</th>
    <th>Year</th>
    <th>Semester</th>
      <th>Grade</th>
   <th>Address</th>
   </tr>
   <xsl:for-each select="student-details/student">
   <xsl:sort select="grade" />
   <tr bgcolor="#f0c0a0">
    <td><xsl:value-of select="name" /></td>
    <td><xsl:value-of select="reg_no" /></td>
    <td><xsl:value-of select="year" /></td>
    <td><xsl:value-of select="semester" /></td>
    <td><xsl:value-of select="grade" /></td>
    <td><xsl:value-of select="address" /></td>
   </tr>
```

```
    </xsl:for-each>
    </table>
<hr/>
<h1>First Year Students</h1>
    <table border="1">
    <tr bgcolor="white">
      <th>Name</th>
      <th>Register No</th>
      <th>Year</th>
      <th>Semester</th>
        <th>Grade</th>
     <th>Address</th>
    </tr>

    <xsl:for-each select="student-details/student[year = 1]">
    <tr bgcolor="#f0c0a0">
      <td><xsl:value-of select="name"/></td>
      <td><xsl:value-of select="reg_no"/></td>
      <td><xsl:value-of select="year"/></td>
      <td><xsl:value-of select="semester"/></td>
      <td><xsl:value-of select="grade"/></td>
      <td><xsl:value-of select="address"/></td>
    </tr>
    </xsl:for-each>
    </table>
<hr />

<h1><center>Students with "A" Grade</center></h1>
    <table border="1">
      <tr bgcolor="white">
        <th>Student Name</th>
            <th>Register Number</th>
            <th>Year</th>
            <th>Semester</th>
            <th>Grade</th>
            <th>Address</th>
      </tr>
    <xsl:for-each select="student-details/student[grade = 'A']">
      <tr bgcolor="#f0c0a0">
            <td><xsl:value-of select="name"/></td>
            <td><xsl:value-of select="reg_no"/></td>
            <td><xsl:value-of select="year"/></td>
```

```
            <td><xsl:value-of select="semester"/></td>
            <td><xsl:value-of select="grade"/></td>
            <td><xsl:value-of select="address"/></td>
          </tr>
        </xsl:for-each>
        </table>

      </center>
    </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

OUTPUT:



PART A

1. **What is PHP? (NOV/DEC 2016)**

Hypertext Preprocessor is an acronym for PHP. PHP is an open-source server scripting language, and a powerful tool for creating dynamic and interactive Web pages. PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

2. **What is a PHP file?**
   - PHP files can contain text, HTML, CSS, JavaScript, and PHP code
   - PHP code are executed on the server, and the result is returned to the browser as plain HTML
   - PHP files have extension ".php"

3. **List out the features of PHP.**
   - PHP can generate dynamic page content
   - PHP can create, open, read, write, delete, and close files on the server
   - PHP can collect form data
   - PHP can send and receive cookies
   - PHP can add, delete, modify data in your database
   - PHP can be used to control user-access
   - PHP can encrypt data
   - With PHP we are not limited to output HTML. we can output images, PDF files, and even Flash movies. we can also output any text, such as XHTML and XML.

4. **What are advantages of PHP?**
   - Used for creating dynamic and interactive web pages
   - PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
   - PHP is compatible with almost all servers used today (Apache, IIS, etc.)
   - PHP supports a wide range of databases
   - PHP is free. Download it from the official PHP resource
   - PHP is easy to learn and runs efficiently on the server side

5. **Write the syntax for writing PHP scripts.**

A PHP script can be placed anywhere in the document.
A PHP script starts with **<?php** and ends with **?>**:

```
<?php
    // PHP code goes here
?>
```

The default file extension for PHP files is ".php".
A PHP file normally contains HTML tags, and some PHP scripting code.

6. **Write the PHP script to print "Hello" message on the screen.**

```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>

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

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

7. How will you create variables in PHP?

Variables are "containers" for storing information.

In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example:

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

8. List the rules for creating variables in PHP.

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

9. What is meant by variable scope?

The lifetime of a variable during the program execution is called the scope of a variable and it is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- Local - A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function
- global - A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function
- static – A variable declared inside the function with the keyword "static" and it helps to preserve the variable even after the function execution has completed.

10. List the various data types supported in PHP.

PHP variables are loosely typed – they can contain different types of data at different times. The various data types supported in PHP are:

Type	Description
int, integer	Whole numbers (i.e., numbers without a decimal point)
float, double, real	Real numbers (i.e., numbers containing a decimal point)
String	Test enclosed in either single(' ') or double (" ") quotes.
bool, boolean	True or false
Array	Group of elements
Object	Group of associated data and methods
Resource	An external resource – usually information form database
NULL	No Value

11. Name some built-in functions in PHP. (NOV / DEC 2015)

- ✓ Array Functions
- ✓ Class/Object Functions
- ✓ Character Functions
- ✓ Date & Time Functions

- ✓ Error Handling Functions
- ✓ MySQL Functions
- ✓ ODBC Functions
- ✓ String Functions
- ✓ XML Parsing Functions

12. How will create user-defined function in PHP?

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads.

A function will be executed by a call to the function.

Creating User Defined Function in PHP

A user defined function declaration starts with the word "function":

Syntax

```
function functionName()
{
    code to be executed;
}
```

13. What is form validation PHP? (APRIL/MAY 2021, 2022)

Form validation is a "technical process where a web-form checks if the information provided by a user is correct." The form will either alert the user that they messed up and need to fix something to proceed, or the form will be validated and the user will be able to continue with their registration process.

There are two types of validation are available in PHP. They are as follows –

¬ Client-Side Validation – Validation is performed on the client machine web browsers.

¬ Server Side Validation – After submitted by data, The data has sent to a server and perform validation checks in server machine.

Some of Validation rules for field:

Field	Validation Rules
Name	Should require letters and white-spaces
Email	Should require @ and .
Website	Should require a valid URL
Radio	Must be selectable at least once
Check Box	Must be checkable at least once
Drop Down menu	Must be selectable at least once

14. Write a PHP program to add two numbers.

```
<html>
<head>
<title>Writing PHP Function to add Two Numbers</title>
</head>
<body>
<?php
        function addFunction($num1, $num2)
        {
          $sum = $num1 + $num2;
          echo "Sum of the two numbers is : $sum";
        }
        addFunction(10, 20);
?>
```

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

Output:

Sum of the two numbers is : 30

15. Write the steps to connect a PHP program with a database.

i. Open a connection to MySQL itself - **mysql_connect()**
ii. Specify the database we want to open - **mysql_select_db()**
iii. Close the connection - **mysql_close()**

16. How to set cookies in PHP?

PHP provided **setcookie()** function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

Syntax: setcookie(name, value, expire, path, domain, security);

Example:

```
<?php
  setcookie("name", "John Watkin", time()+3600, "/","", 0);
  setcookie("age", "36", time()+3600, "/", "",  0);
?>
```

17. What is meant by Regular Expression?

Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.

Using regular expression we can search a particular string inside another string, we can replace one string by another string and you can split a string into many chunks.

18. What are the two sets of regular expressions offered by PHP?

PHP offers functions specific to two sets of regular expression functions:

- POSIX extended Regular Expressions
- PERL compatible Regular Expressions

19. List some POSIX regular expression functions.

Function	Description
ereg()	The ereg() function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise.
ereg_replace()	The ereg_replace() function searches for string specified by pattern and replaces pattern with replacement if found.
eregi()	The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive.
eregi_replace()	The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.
split()	The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
spliti()	The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive.

sql_regcase()	The sql_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

20. List some Perl compatible regular expression functions.

Function	Description
preg_match()	The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.
preg_match_all()	The preg_match_all() function matches all occurrences of pattern in string.
preg_replace()	The preg_replace() function operates just like ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.
preg_split()	The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
preg_grep()	The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
preg_quote()	Quote regular expression characters

21. What is XML?

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language for describing the format for data exchanged between applications over the internet. XML permits document authors to create new markup languages for describing any type of data.

22. List the important characteristics of XML.

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

23. Mention the uses (advantages) of XML.

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

24. What are the XML Syntax Rules?

1. The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
2. If document contains XML declaration, then it strictly needs to be the first statement of the XML document.

3. ttribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and *href* are considered two different XML attributes.
4. Same attribute cannot have two values in a syntax. The following example shows incorrect syntax because the attribute *b* is specified twice:
5. ....
6. Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks. Following example demonstrates incorrect xml syntax:
7. ....
8. In the above syntax, the attribute value is not defined in quotation marks.
9. The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case.
10. To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files.
11. Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored.
12. Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly.

25. What do you mean by XML declaration? Give an example. (MAY/JUNE 2013)

The XML declaration is a *processing instruction* that identifies the document as being XML. All XML documents should begin with an XML declaration.

For example,

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

26. What is meant by a XML namespace?(April / May 2011, 2014, Nov / Dec 2012, 2013,NOV/DEC 2016)

An XML namespace is a collection of element and attribute names. XML namespace provide a means for document authors to unambiguously refer to elements with the same name (i.e., prevent collisions).

Example:

<subject>Geometry</subject> // data for student from school

and

<subject>Cardiology</subject> // data for student from medicine

Both the markup uses the element “subject” to markup data. Namespace can differentiate these two “subject” elements:

<highschool:subject>Geometry</highschool:subject>

and **<medicalschool:subject>Cardiology</medicalschool:subject>**

Both **highschool** and **medicalschool** are namespace prefixes.

27. How the namespace is declared?

A Namespace is declared using reserved attributes. Such an attribute name must either be **xmlns** or begin with **xmlns:** shown as below:

<element xmlns:name="URL">

Syntax:

- The Namespace starts with the keyword **xmlns**.
- The word **name** is the Namespace prefix.
- The **URL** is the Namespace identifier.

Example:

```
<root>
<highschool:subject xmlns:highschool="http://www.abcschool.edu/subjects">
 Geometry
</highschool:subject>
<medicalschool:subject xmlns:medicalshool="http://www.rmc.org/subjects">
Cardiology
</medicalschool:subject>
</root>
```

28. What is meant by Document Type Definition? (NOV/DEC 2022)

Rules that define the legal elements and attributes for XML documents are called Document Type Definitions (DTD) or XML Schemas.

There are two different document type definitions that can be used with XML:

- DTD - The original Document Type Definition
- XML Schema - An XML-based alternative to DTD

29. What is the purpose of DTD?

The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements:

Example DTD :

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

<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget to complete the work !</body>
< </note>
```


The DTD above is interpreted like this:

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

30. What is meant by Internal DTD?

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means, the declaration works independent of external source.

Syntax

The syntax of internal DTD is as shown:

<!DOCTYPE root-element [element-declarations]>

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
  <!ELEMENT address (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

31. Give an example for External DTD.

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes. To refer it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax:

Following is the syntax for external DTD:

<!DOCTYPE root-element SYSTEM "file-name">

where *file-name* is the file with *.dtd* extension.

Example:

The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

32. What is XML Schema? (April/May 2021)

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

33. Give the syntax to declare XML schema.

Syntax:

The common syntax to declare a schema in our XML document as follows:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

Example: The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="company" type="xs:string" />
      <xs:element name="phone" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

34. What are the data types supported by XML Schema?

Type	Description	Example
String	A character string	"hello"
Boolean	True or False	true
Decimal	A decimal numeral	5, -12, -45.78
Float	A floating-point number	0, 12, -109.75, NaN
Double	A floating-point number	0, 12, -109.75, NaN
Long	A whole number	1234567890, -1234567890
Int	A whole number	12345, -12345
Short	A whole number	12, -345
Date	A date consisting of year, month and day	2005-05-10
Time	A time consisting of hours, minutes and seconds	16:30:25-05:00

35. What is meant by XML DOM?

The W3C Document Object Model (DOM) is a platform and language-neutral Application Programming Interface (API) that allows programs and scripts to dynamically access and update the content, structure, and style of a HTML, XHTML & XML document.

36. Define DOM tree.

In XML document the information is maintained in hierarchical structure, this hierarchical structure is referred as ***Node Tree*** or ***DOM Tree.***

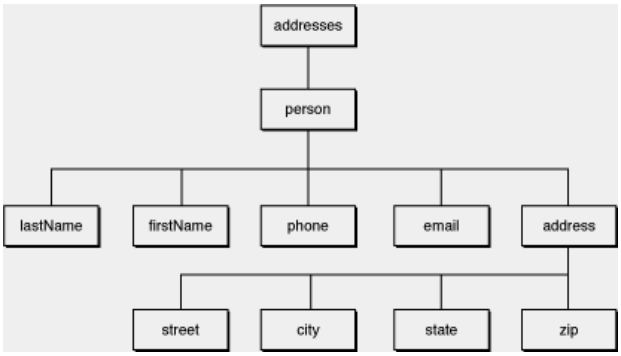
The structure of the node tree begins with the root element and spreads out to the child elements till the lowest level.

The nodes in the node tree have a hierarchical relationship to each other.The terms parent, child, and sibling are used to describe the relationships. Parent nodes have children. Children on the same level are called siblings (brothers or sisters).

- In a node tree, the top node is called the root
- Every node, except the root, has exactly one parent node
- A node can have any number of children
- A leaf is a node with no children
- Siblings are nodes with the same parent

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE addresses SYSTEM "addresses.dtd">
<addresses>
   <person idnum="0123">
      <lastName>Doe</lastName>
      <firstName>John</firstName>
      <phone location="mobile">(201) 345-6789</phone>
      <email>jdoe@foo.com</email>
      <address>
         <street>100 Main Street</street>
         <city>Somewhere</city>
         <state>New Jersey</state>
         <zip>07670</zip>

      </address>
   </person>
</addresses>
```

The following tree of element nodes represents this document:



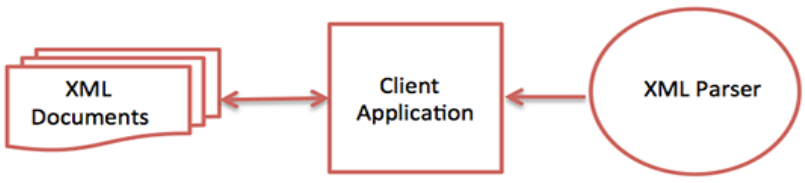
37. What is meant by DOM Parser?

An XML parser that creates the DOM tree structure is known as **DOM parser.**

38. What is an XML Parser?

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers.

Following diagram shows how XML parser interacts with XML document:



The goal of a parser is to transform XML into a readable code.

39. Explain the use of XML Parser. (APRIL/MAY 2022)

✓ It checks for proper format of the XML document (Well Formed Document) and may also validate the XML documents (Valid XML Document).

✓ XML parser validates the document and check that the document is well formatted.

40. What is meant by XML validation?

XML validation is the process of checking a document written in XML (eXtensible Markup Language) against its specified DTD to confirm that it is both well-formed and also "valid". When a document fails to conform to a DTD or a schema, the validator displays an error message.

41. What is meant by well-formed XML document?

An XML document with correct syntax is "Well Formed". A well-formed XML document must follow the following XML syntax rules:

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quote.

Example well-formed XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget to complete the work!</body>
</note>
```


42. What is meant by "Valid XML document"?

An XML document that contains the proper elements with the proper attributes in the proper sequence according to the DTD specification is called a "Valid" XML document. An XML document cannot be valid unless it is well-formed.

43. What are the two types of XML parsers? And differentiate them.(U/AN)

Two types of XML Parsers:

1. Non-Validating XML Parsers
2. Validating XML Parsers

Non-Validating XML Parsers	Validating XML Parsers
A non-validating parser checks if a document follows the XML syntax rules. It builds a tree structure from the tags used in XML document and return an error only when there is a problem with the syntax of the document.	A Validating parser checks the syntax, builds the tree structure, and compares the structure of the XML document with the structure specified in the DTD associated with the document.
Non validating parsers process a document faster because they do not have to check every element against a DTD. In other words, these parsers check whether an XML document adheres to the rules of well-formed document.	Validating parsers are slower because, in addition to checking whether an XML document is well formed; validating parsers also check whether the XML document adheres to the rules in the DTD used by the XML document.
The Expat parser is an example of non-validating parser.	Microsoft MSXML parser is an example of a validating parser.

44. Define XSL.

The **Extensible Style sheet Language (XSL)** is an XML vocabulary that decides how an XML document data should look on the web browser. XSL is group of three technoliges:

1. **XSL Transformations (XSLT):** is technology for transforming the structure of the XML document data into another structure.
2. **XML Path Language (XPath):** is a string-based language of expressions which defines the syntax and semantics for efficiently locating elements and attributes in XML documents.
3. **XSL Formatting Objects (XSL-FO):** It is a separate XML vocabulary for defining document style properties of an XML document .(print-oriented)

45. Explain two types of XSL information.

An XSL document normally contains **two types of information**:

- ✓ **Template data** - Which is text that is copied to the output XML text document with little or no change? i.e. Everything in the body of the document that is not XSL markup is *template data* And
- ✓ **XSL markup** - which controls the transformation process always **start with "xsl:".**

46. Name any four XSL elements .Mentions its use.

a. The **<xsl:variable>** XSL element creates a named variable.
b. The **<xsl:value-of>** XSL element processes a value.
c. The **<xsl:if>** XSL element invokes the enclosed code if the condition specified by the test attribute is true.
d. The **<xsl:sort>** XSL element sorts a list of elements, such as within <xsl:for-each>.
e. The **<xsl:template>** XSL element encapsulates a segment of XSL code, similar to a method, procedure, or function in other languages.

47. What is XSLT?

XSLT stands for XSL Transformations. XSLT is a language for transforming XML documents into XHTML documents or to other XML documents. XSLT uses XPath to navigate in XML documents. With XSLT user can add/remove elements and attributes to or from the output file.

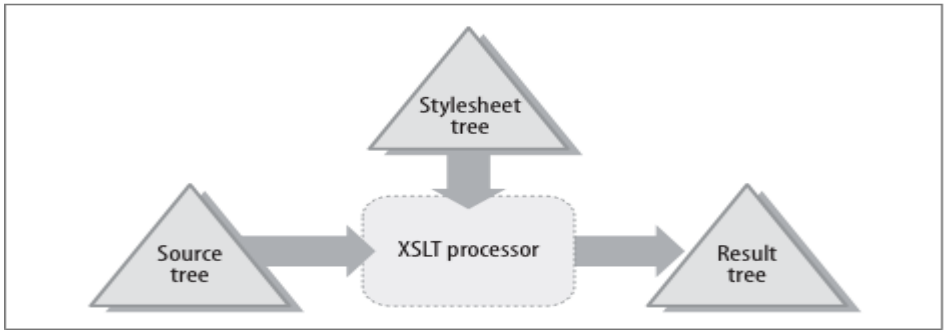
48. What is XPath?

XPath is a language used to navigate through elements and attributes in an XML document.

- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT
- XPath is a W3C recommendation

49. How does XSLT works?

In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.



50. Why is XSLT an important tool in development of web applications?
(MAY/JUNE 2016)

XSLT is the most important part of XSL.

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

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

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

51. When should the super global arrays in PHP be used? Which super global array in PHP would contain a HTML form's POST data? (MAY/JUNE 2016)

PHP super global variable is used to access global variables from anywhere in the PHP script.

52. List the important characteristics of PHP. (NOV/DEC 2018)

- Familiarity
- Simplicity
- Efficiency
- Security
- Flexibility
- Open source
- Object Oriented

53. Explain DTD for XML Schemas. (NOV/DEC 2018)

A DTD is a Document Type Definition. A DTD defines the structure and the legal elements and attributes of an XML document. An XML Schema describes the structure of an XML document. XML document has a reference to a DTD.

54. Give different ways to validate a form (NOV/DEC 2022)

There are two different types of form validation – Client side validation and Server side validations.

PART – B

1. Discuss in detail about how to create and use variables in PHP with example program.
2. What are the different data types available in PHP? Explain about converting between different data types with example program.
3. Write a PHP program using arithmetic operator.
4. Explain the features of built-in functions in PHP with examples. **(APRIL/MAY 2022)**
5. Write a PHP program to do string manipulation. **(NOV / DEC 2015)**
6. Explain how user-defined functions are created in PHP.
7. Explain how input from an XHTML form is received in a PHP program.
8. How will you connect a PHP program with database? Explain with example application.
9. Explain how cookies are handled in PHP.
10. Explain in detail about using Regular Expressions in PHP.
11. Explain XML DTD. **(APRIL/MAY 2022)**
12. What are different types of DTD? Explain the same with example.
13. Explain the role of XML name spaces with examples. **(MAY/JUNE 2012)**
14. What is XML Schema? Explain how to create and use XML Schema document.

(NOV/DEC 2015)

15. Explain how a XML document can be displayed on a browser. **(APRIL/MAY 2011)**

16. Explain Document Tree with an example. **(MAY/JUNE 2011, MAY/JUNE 2012)**

17. Explain in detail about XSL. **(NOV/ EC 2013)**

18. Give an XSLT document and a source XML document explain the XSLT transformation process that produces a single result XML document. **(NOV/DEC 2012)**

19. Explain in detail about XML parsers and validation. **(NOV/DEC 2015, MAY/JUNE 2016)**

20. Create a webserver based chat application using PHP. The application should provide the function functions.

- Login
- Send message (to one or more contacts)
- Receive messages(from one or more contacts)
- Add/delete /modify contact list of the user

21. Discuss the application's user interface and use comments in PHP to explain the code clearly. **(MAY/JUNE 2016)**

22. List at least five significant differences between DID and XML schema for defining XML document structures with appropriate examples. **(MAY/JUNE 2016)**

23. Explain the string comparison capability of PHP using regular expressions with an example. **(NOV/DEC2016)**

24. Explain the steps in connecting a PHP code to a database. (NOV/DEC2016)

25. Explain in detail the XML schema, built in and user defined data type in detail. **(NOV/DEC2016)**

26. Design simple calculator using PHP. (NOV/DEC2018)

27. Explain about the controls statements in PHP with example. (NOV/DEC 2018)

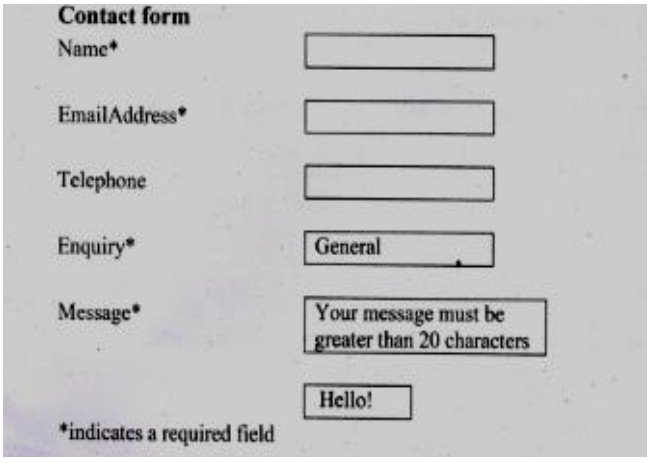
28. Design a PHP application for College Management System with appropriate built-in functions and database. **(NOV/DEC 2018)**

29. What is spring framework ? What is use of it ? What are the advantages of Spring Framework? **(APRIL/MAY 2021)**

30. Using XSLT, how would you extract a specific attribute from an element in an Xml document? When constructing an XML DTD, how do you create an external entity reference in an attribute value ? **(APRIL/MAY 2021)**

31. Write the HTML code for creating a feedback form as shown below. Include comments in code to highlight the markup elements and their purpose. The HTML form should use POST for submitting the form to a program ProcessContactForm.PHP. **(MAY/JUNE 2016)**

Internet Programming



35. Illustrate the steps and procedure to connect the PHP to any database. write a program of your own demonstrate **(NOV-DEC 2022)**

36. Write a program using PHP to check whether the input to a function calculating factorial is valid(>0) or not**(NOV-DEC 2022)**

UNIT V INTRODUCTION TO ANGULAR and WEB APPLICATIONS FRAMEWORKS

Introduction to AngularJS, MVC Architecture, Understanding ng attributes, Expressions and data binding, Conditional Directives, Style Directives, Controllers, Filters, Forms, Routers, Modules, Services; Web Applications Frameworks and Tools – Firebase- Docker- Node JS- React- Django- UI & UX.

INTRODUCTION TO ANGULARJS

- AngularJS is a popular open-source JavaScript framework developed and maintained by Google.
- It is primarily used for building dynamic web applications and is known for its capability to create single-page applications (SPAs).
- AngularJS follows the Model-View-Controller (MVC) architectural pattern, which promotes a clear separation of concerns.
- AngularJS is a JavaScript **framework**. It is a client side scripting framework. It can be added to an HTML page with a <script> tag.
- AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.
- AngularJS is a JavaScript framework written in JavaScript and it is based on MVC **Architecture**.
- AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js
"></script>
```

ANGULARJS EXTENDS HTML

AngularJS extends HTML with **ng-directives**.

The **ng-app** directive defines an AngularJS application.

The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-bind** directive binds application data to the HTML view.

ANGULAR JS EXAMPLE:

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js
"></script>
<body>
<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>
</body>
</html>
```


OUTPUT:

Input something in the input box:

Name:

Example explained:

AngularJS starts automatically when the web page has loaded.

The **ng-app** directive tells AngularJS that the element is the "owner" of an AngularJS **application**.

The **ng-model** directive binds the value of the input field to the application variable **name**.

The **ng-bind** directive binds the content of the <p> element to the application variable **name**.

ANGULARJS APPLICATIONS

- It is mainly used for creating single web page application.

- AngularJS **modules** define AngularJS applications.
- AngularJS **controllers** control AngularJS applications.
- The **ng-app** directive defines the application, the **ng-controller** directive defines the controller.

MVC ARCHITECTURE

What is MVC?

The **Model-View-Controller (MVC)** framework is an architectural/design pattern that separates an application into three main logical components **Model**, **View**, and **Controller**. Each architectural component is built to handle specific development aspects of an application.

It isolates the business logic and presentation layer from each other. It was traditionally used for desktop **graphical user interfaces (GUIs)**.

Nowadays, MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects.

It is also used for designing mobile apps.

MVC was created by **Trygve Reenskaug**.

The main goal of this design pattern was to solve the problem of users controlling a large and complex data set by splitting a large application into specific sections that all have their own purpose.

Features of MVC :

- It provides a clear separation of **business logic, Ul logic, and input logic.**
- It offers full control over your HTML and URLs which makes it easy to design web application architecture.
- It is a powerful URL-mapping component using which we can build applications that have comprehensible and searchable URLs.
- It supports **Test Driven Development (TDD).**

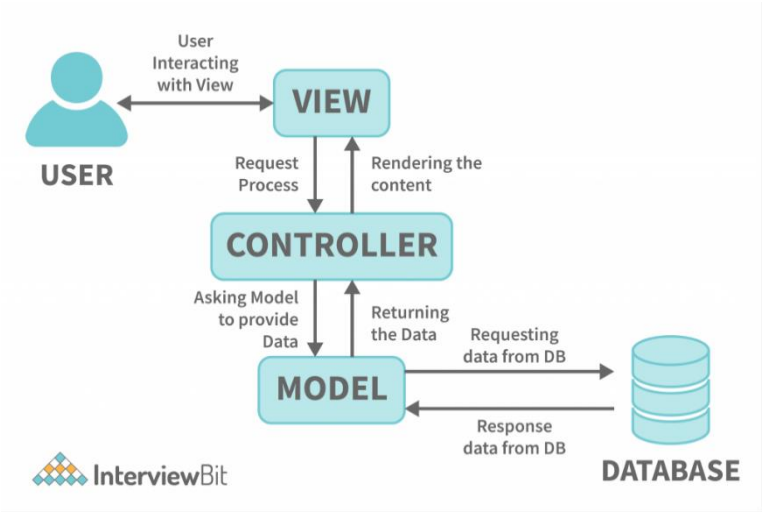
Components of MVC :

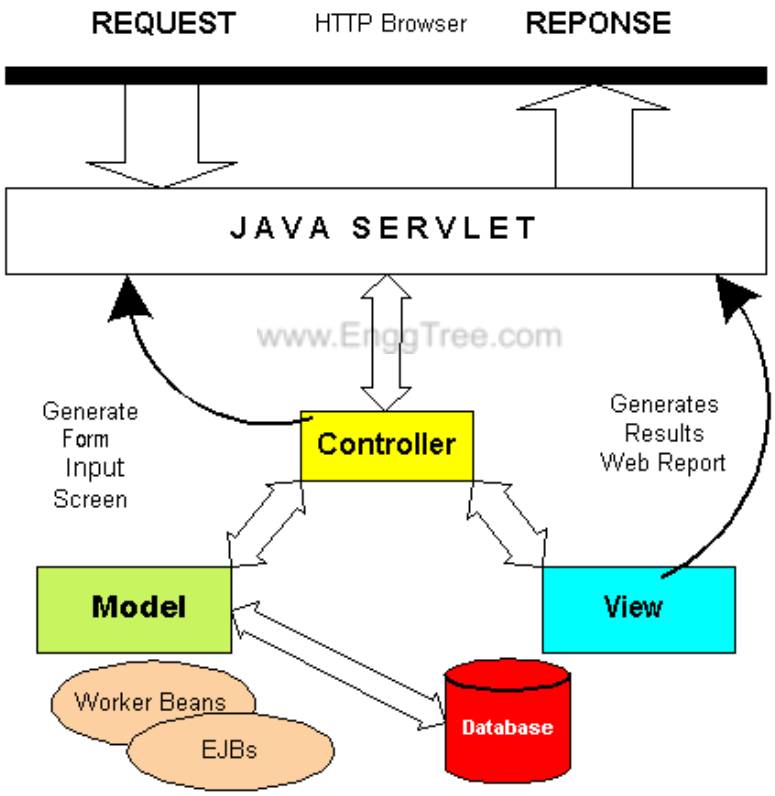
The MVC framework includes the following 3 components:

- Controller
- Model
- View

Controller:

The controller is the component that enables the interconnection between the views and the model so it acts as an intermediary. The controller doesn't have to worry about handling data logic, it just tells the model what to do. It processes all the business logic and incoming requests, manipulates data using the **Model** component, and interact with the **View** to render the final output.





View:

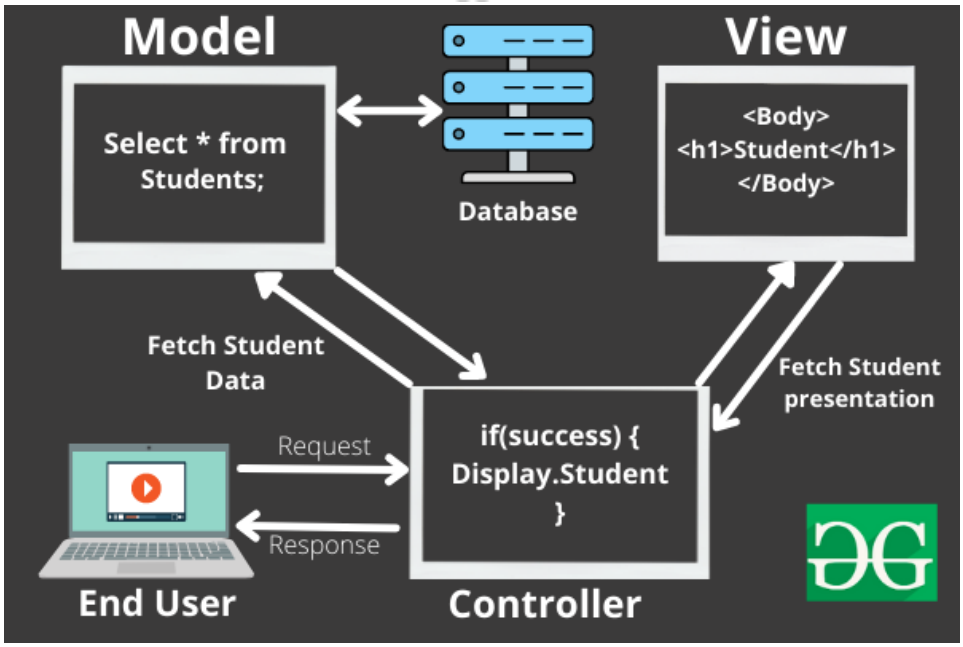
The **View** component is used for all the UI logic of the application. It generates a user interface for the user. Views are created by the data which is collected by the model component but these data aren't taken directly but through the controller. It only interacts with the controller.

Model:

The **Model** component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. It can add or retrieve data from the database. It responds to the controller's request because the controller can't interact with the database by itself. The model interacts with the database and gives the required data back to the controller.

Working of the MVC framework with an example:

Let's imagine an end-user sends a request to a server to get a list of students studying in a class. The server would then send that request to that particular controller that handles students. That controller would then request the model that handles students to return a list of all students studying in a class.



The flow of data in MVC Components

The model would query the database for the list of all students and then return that list back to the controller. If the response back from the model was successful, then the controller would ask the view associated with students to return a presentation of the list of students. This view would take the list of students from the controller and render the list into HTML that can be used by the browser.

The controller would then take that presentation and returns it back to the user. Thus ending the request. If earlier the model returned an error, the controller would handle that error by asking the view that handles errors to render a presentation for that particular error. That error presentation would then be returned to the user instead of the student list presentation.

As we can see from the above example, the model handles all of the data. The view handles all of the presentations and the controller just tells the model and view of what to do. This is the basic architecture and working of the MVC framework.

The MVC architectural pattern allows us to adhere to the following design principles:

1. **Divide and conquer.** The three components can be somewhat independently designed.
2. **Increase cohesion.** The components have stronger layer cohesion than if the view and controller were together in a single UI layer.
3. **Reduce coupling.** The communication channels between the three components are minimal and easy to find.
4. **Increase reuse.** The view and controller normally make extensive use of reusable components for various kinds of UI controls. The UI, however, will become application-specific, therefore it will not be easily reusable.
5. **Design for flexibility.** It is usually quite easy to change the UI by changing the view, the controller, or both.

Advantages of MVC:

- Codes are easy to maintain and they can be extended easily.
- The MVC **model** component can be tested separately.
- The components of MVC can be developed simultaneously.
- It reduces complexity by dividing an application into three units. **Model, view, and controller.**
- It supports **Test Driven Development (TDD).**
- It works well for Web apps that are supported by large teams of web designers and developers.
- This architecture helps to test components independently as all classes and objects are independent of each other
- **Search Engine Optimization (SEO)** Friendly.

Disadvantages of MVC:

- It is difficult to read, change, test, and reuse this model
- It is not suitable for building small applications.
- The inefficiency of data access in view.
- The framework navigation can be complex as it introduces new layers of abstraction which requires users to adapt to the decomposition criteria of MVC.
- Increased complexity and Inefficiency of data

Popular MVC Frameworks:

Some of the most popular and extensively used MVC frameworks are listed below.

- Ruby on Rails
- Django
- CherryPy
- Spring MVC
- Catalyst

- Rails
- Zend Framework
- Fuel PHP
- Laravel
- Symphony

MVC is generally used on applications that run on a single graphical workstation. The division of logical components enables readability and modularity as well as it makes it more comfortable for the testing part.

UNDERSTANDING NG ATTRIBUTES

AngularJS Directives

AngularJS facilitates you to extend HTML with new attributes. These attributes are called directives. AngularJS directives are HTML attributes with an **ng** prefix.

There is a set of built-in directive in AngularJS which offers functionality to your applications. You can also define your own directives.

Directives are special attributes starting with ng- prefix. Following are the most common directives:

- ng-app: This directive starts an AngularJS Application.
- ng-init: This directive initializes application data.
- ng-model: This directive
- ng-model: This directive defines the model that is variable to be used in AngularJS.
- ng-repeat: This directive repeats html elements for each item in a collection.

AngularJS Directives List

AnglarJS directives are used to add functionality to your application. You can also add your own directives for your applications.

Following is a list of AngularJS directives:

Directive	Description
ng-app	It defines the root element of an application.
ng-bind	It binds the content of an html element to application data.
ng-bind-html	It binds the inner HTML of an HTML element to application data, and also removes dangerous code from the html string.
ng-bind-template	It specifies that the text content should be replaced with a template.
ng-blur	It specifies a behavior on blur events.
ng-change	It specifies an expression to evaluate when content is being changed by the user.
ng-checked	It specifies if an element is checked or not.
ng-class	It specifies css classes on html elements.
ng-class-even	It is same as ng-class, but will only take effect on even rows.
ng-class-odd	It is same as ng-class, but will only take effect on odd

	rows.
ng-click	It specifies an expression to evaluate when an element is being clicked.
ng-cloak	It prevents flickering when your application is being loaded.
ng-controller	It defines the controller object for an application.
ng-copy	It specifies a behavior on copy events.
ng-hide	It hides or shows html elements.
ng-cut	It specifies a behavior on cut events.
ng-if	It removes the html element if a condition is false.
ng-focus	It specifies a behavior on focus events.
ng-init	It defines initial values for an application.
ng-href	It specifies a URL for the <a> element.
ng-keydown	It specifies a behavior on keydown events.
ng-keypress	It specifies a behavior on keypress events.

ng-keyup	It specifies a behavior on keyup events.
ng-jq	It specifies that the application must use a library, like jQuery.
ng-include	It includes html in an application.
ng-list	It converts text into a list (array).
ng-options	It specifies <options> in a <select> list.
ng-paste	It specifies a behavior on paste events.
ng-show	It shows or hides html elements.
ng-src	It specifies the src attribute for the <img> element.
ng-submit	It specifies expressions to run on onsubmit events.
ng-style	It specifies the style attribute for an element.
ng-repeat	It defines a template for each data in a collection.
ng-switch	It specifies a condition that will be used to show/hide child elements.
ng-value	It specifies the value of an input element.

ng-disabled	It specifies if an element is disabled or not.
ng-form	It specifies an html form to inherit controls from.
ng-model	It binds the value of html controls to application data.

ng-app directive

ng-app directive defines the root element. It starts an AngularJS Application and automatically initializes or bootstraps the application when web page containing AngularJS Application is loaded. It is also used to load various AngularJS modules in AngularJS Application.

SYNTAX

***<div** ng-app = ""> ... ** ***

' : This is the basic syntax of the ng-app directive. It is placed on an HTML element (here, a) to designate it as the root of the AngularJS application named 'myApp'.

EXAMPLE:

```
<!DOCTYPE html>
<html>
<head>
   <title>AngularJS App</title>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></scr
ipt>
</head>
<body>
```

```
<!-- The ng-app directive defines the root element of the AngularJS application
-->
<div ng-app="myApp">
  <h1>Welcome to my AngularJS App!</h1>
  <!-- Other AngularJS directives and content -->
</div>
<script>
  // AngularJS module creation
  var app = angular.module('myApp', []);
  // Define controllers, services, etc., within this module
</script>
</body>
</html>
```

- The HTML file includes the AngularJS library through a script tag.
- Inside a with the ng-app="myApp" attribute, you can place AngularJS content and directives.
- This example simply displays a welcoming header within the AngularJS root element.
- The script section defines an AngularJS module named 'myApp' with an empty dependency array [].

ng-init directive

The ng-init directive evaluates the given expression. This expression can be used to initialize variables, set up event listeners, or perform other tasks that need to be done when the application starts.

SYNTAX:

<any ng-init="expression"> ...</any>

The expression can be any JavaScript expression. The expression will be evaluated in the current scope, and the result of the expression will be assigned to the specified variable.

EXAMPLE:

Here is an example of how to use the ng-init directive to initialize a variable:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<body>
<div ng-app="" ng-init="myText='Hello World!'">
<h1>{{myText}}</h1>
<p>The ng-init directive has created an AngularJS variable, which you can use
in the application.</p>
</div>
</body>
</html>
```


OUTPUT:
Hello World!
The ng-init directive has created an AngularJS variable, which you can use in the application.

ng-model directive

The ng-model directive is a core Angular directive that binds input, select, and textarea elements to application data.

It establishes a two-way data binding between the view (HTML) and the model (TypeScript), ensuring that any changes made to the input field are reflected in the corresponding data property and vice versa.

In following example, we've defined a model named "name".

SYNTAX:

The syntax for the ng-model directive is as follows:

```
<input type="text" ng-model="propertyName">
<select ng-model="propertyName">
  <option value="option1">Option 1</option>
  <option value="option2">Option 2</option>
</select>
<textarea ng-model="propertyName">Initial Text</textarea>
```

EXAMPLE:

Consider a component with a property named userName:

TYPESCRIPT

```
export class MyComponent {
  userName: string = '';
}
```


To bind an input field to this property, use the ng-model directive as follows:

HTML

```
<input type="text" ng-model="userName">
```

Any changes made to the input field will be reflected in the userName property, and any changes to the userName property will be reflected in the input field.

ng-repeat directive

ng-repeat directive repeats html elements for each item in a collection. In following example, we've iterated over array of countries.

SYNTAX:

```
<div ng-app="myApp">
   <ul>
```

```
    <li ng-repeat="item in items">{{ item }}</li>
  </ul>
</div>
```

**<li ng-repeat="fruit in fruits">{{ fruit }}
: This demonstrates the **ng-repeat directive in action. It is applied to an **<li>** element within an AngularJS application and iterates through the **fruits** array, displaying each item in the array.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>AngularJS ng-repeat Example</title>
  <script
```



```
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></scr
ipt>
</head>
<body>
<!-- The ng-app directive defines the root element of the AngularJS application
-->
<div ng-app="myApp">
  <h2>Fruits List:</h2>
  <ul>
    <!-- Using ng-repeat to loop through items -->
    <li ng-repeat="fruit in fruits">{{ fruit }}</li>
  </ul>
</div>
```

```
<script>
    // AngularJS module creation
    var app = angular.module('myApp', []);

    // Define controllers, services, etc., within this module
    app.controller('myController', function($scope) {
        $scope.fruits = ['Apple', 'Banana', 'Orange', 'Grapes', 'Mango'];
    });
</script>
</body>
</html>
```

- The HTML file includes the AngularJS library using a script tag.
- Inside the AngularJS root element (**ng-app="myApp"**), there is an unordered list (**<ul>**) containing list items (**<li>**).
- **ng-repeat="fruit in fruits"** creates a loop that iterates through the **fruits** array.
- For each item in the **fruits** array, the **{{ fruit }}** expression displays the item's value within an **<li>** element.
- The AngularJS script defines an **app** module and a controller named **myController**.
- The controller sets the value of **$scope.fruits** to an array of fruit names.

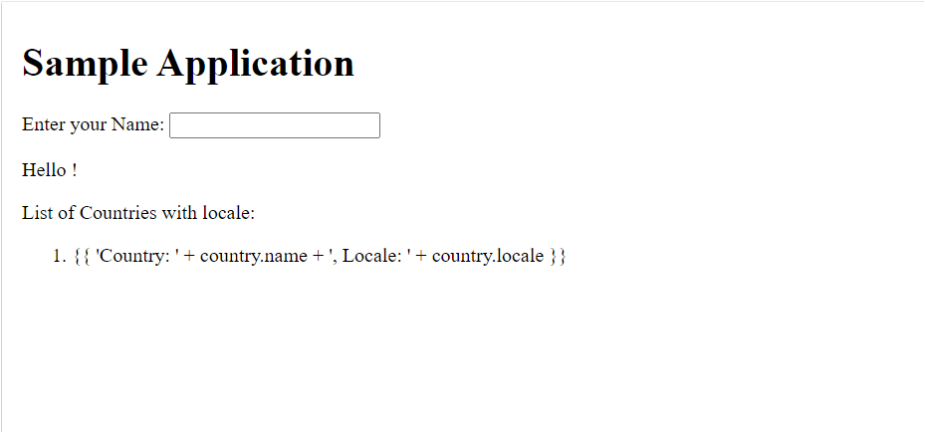
AngularJS directives Example

Let's take an example to use all the above discussed directives:

```
<!DOCTYPE html>
<html>
<head>
    <title>AngularJS Directives</title>
```

```
</head>
<body>
    <h1>Sample Application</h1>
<div ng-app = "" ng-init = "countries = [{locale:'en-IND',name:'India'}, {locale:'en-PAK',name:'Pakistan'}, {locale:'en-AUS',name:'Australia'}]">
      <p>Enter your Name: <input type = "text" ng-model = "name"></p>
      <p>Hello <span ng-bind = "name"></span>!</p>
      <p>List of Countries with locale:</p>
          <ol>
        <li ng-repeat = "country in countries">
          {{ 'Country: ' + country.name + ', Locale: ' + country.locale }}
        </li>
      </ol>
    </div>
<script src = "http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"
></script>
</body>
</html>
```

OUTPUT:



ANGULARJS EXPRESSIONS

In AngularJS, expressions are used to bind application data to HTML. AngularJS resolves the expression, and return the result exactly where the expression is written.

Expressions are written inside double braces {{expression}}.They can also be written inside a directive:

ng-bind="expression".

AnularJS expressions are very similar to JavaScript expressions. They can contain literals, operators, and variables. For example:

{{ 5 + 5 }} or {{ firstName + " " + lastName }}

AngularJS Expressions Example

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
/script>
<body>
<div ng-app>
<p>A simple expression example: {{ 5 + 5 }}</p>
</div>
</body>
</html>
```


OUTPUT:

A simple expression example: {{ 5 + 5 }}

Note: If you remove the directive "ng-app", HTML will display the expression without solving it.

You can also write expressions wherever you want, AngularJS will resolve the expression and return the result.

AngularJS Numbers

"AngularJS Numbers" refers to the handling, formatting, manipulation, and validation of numeric data within AngularJS applications. It involves using built-in filters, directives like **ng-model**, **ng-change**, and controllers to manage numerical values efficiently, ensuring proper display, input handling, and validation in web applications.

Example:

```
<!DOCTYPE html>
<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="" ng-init="quantity=5;cost=5">
<p>Total in dollar: {{ quantity * cost }}</p>
</div>
</body>
</html>
```

We can use the same example by using ng-bind:

See this example:

```
<!DOCTYPE html>
<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="" ng-init="quantity=5;cost=5">
<p>Total in dollar: <span ng-bind="quantity * cost"></span></p>
```

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

OUTPUT:

Total in dollar: 25

AngularJS Strings

AngularJS Strings" refer to the management, formatting, manipulation, and validation of text data within AngularJS applications. This involves utilizing filters, controllers, directives, and functionalities to handle strings effectively. AngularJS provides tools for transforming string case, limiting string length, manipulating text content, validating input, and facilitating string display through data binding mechanisms.

EXAMPLE

```
<!DOCTYPE html>
<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="" ng-init="firstName='Sonoo';lastName='Jaiswal'">
<p>My full name is: {{ firstName + " " + lastName }}</p>
</div>
</body>
</html>
```

Same example with ng-bind:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
script>
<body>
<div ng-app="" ng-init="firstName='Sonoo';lastName='Jaiswal'">
<p>My full name is: <span ng-bind="firstName + ' ' + lastName"></span></p>
</div>
</body>
</html>
```

OUTPUT:

My full name is: Sonoo Jaiswal

AngularJS Objects

AngularJS Objects" refers to managing, manipulating, and displaying structured data in the form of JavaScript objects within AngularJS applications. This involves utilizing two-way data binding, controllers, expressions, and filters to handle object properties, display their values in the view, and enable interaction with object-based data within web applications

EXAMPLE

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
script>
<body>
<div ng-app="" ng-init="person={firstName:'Sonoo',lastName:'Jaiswal'}">
<p>The name is {{ person.firstName }}</p>
```

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

Same example with ng-bind:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
script>
<body>
<div ng-app="" ng-init="person={firstName:'Sonoo',lastName:'Jaiswal'}">
<p>The name is <span ng-bind="person.firstName"></span></p>
</div>
</body>
</html>
```


OUTPUT:

The name is: Sonoo Jaiswal

AngularJS Arrays

"AngularJS Arrays" involve handling and manipulating collections of data organized in an array format within AngularJS applications. This includes using features like data binding, controllers, directives, and filters to manage array elements efficiently. AngularJS facilitates displaying array elements in the view, performing operations like iteration, filtering, sorting, adding, removing, and updating array items, thereby enabling seamless interaction with array-based data structures in web applications.

EXAMPLE

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
/script>
<body>
<div ng-app="" ng-init="points=[1,15,19,2,40]">
<p>The first result is {{ points[0] }}</p>
</div>
</body>
</html>
```

Same example with ng-bind:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></
/script>
<body>
<div ng-app="" ng-init="points=[1,15,19,2,40]">
<p>The first result is <span ng-bind="points[0]"></span></p>
</div>
</body>
</html>
```

OUTPUT:

The first result is 1

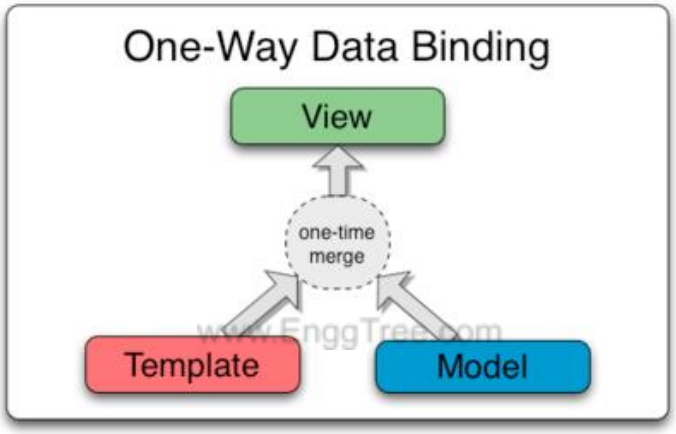
ANGULARJS DATA BINDING

Data binding is a very useful and powerful feature used in software development technologies. It acts as a bridge between the view and business logic of the application.

AngularJS follows Two-Way data binding model.

One-Way Data Binding

The one-way data binding is an approach where a value is taken from the data model and inserted into an HTML element. There is no way to update model

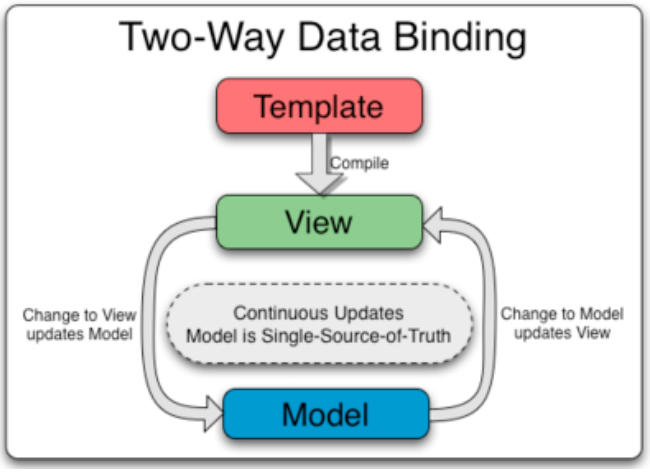


from view. It is used in classical template systems. These systems bind data in only one direction.

Two-Way Data Binding

Data-binding in Angular apps is the automatic synchronization of data between the model and view components.

Data binding lets you treat the model as the single-source-of-truth in your application. The view is a projection of the model at all times. If the model is



changed, the view reflects the change and vice versa.

EXAMPLE PROGRAM:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"><
/script>
<body>
<div ng-app="" ng-init="firstName='Ajeet'">
<p>Input something in the input box:</p>
<p>Name: <input type="text" ng-model="firstName"></p>
<p>You wrote: {{ firstName }}</p>
</div>
</body>
</html>
```

Input something in the input box:

Name: Ajeet

You wrote: Ajeet

OUTPUT:

In the above example, the {{ firstName }} expression is an AngularJS data binding expression. Data binding in AngularJS binds AngularJS expressions with AngularJS data.

{{ firstName }} is bound with ng-model="firstName".

Let's take another example where two text fields are bound together with two ng-model directives:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div data-ng-app="" data-ng-init="quantity=1;price=20">
<h2>Cost Calculator</h2>
Quantity: <input type="number" ng-model="quantity">
Price: <input type="number" ng-model="price">
<p><b>Total in rupees:</b> {{quantity * price}}</p>
</div>
</body>
</html>
```


OUTPUT:

Cost Calculator

Quantity: Price:

Total in rupees: {{quantity * price}}

CONDITIONAL DIRECTIVES IN ANGULARJS

- AngularJS facilitates you to extend HTML with new attributes. These attributes are called directives.
- There is a set of built-in directive in AngularJS which offers functionality to your applications. You can also define your own directives.
- Directives are special attributes starting with ng- prefix.
- Conditional statements of any language are very important. That help to activate functionality based on variable value.
- This angular 4 tutorial help to demonstrate conditional statement with example.

ng-if Directive:

The ng-if Directive in AngularJS is used to remove or recreate a portion of the HTML element based on an expression. The ng-if is different from the ng-hide directive because it completely removes the element in the DOM rather than just hiding the display of the element. If the expression inside it is false then the element is removed and if it is true then the element is added to the DOM.

Used to conditionally render or remove an element based on a boolean expression.

SYNTAX:

```
<div ng-if="condition">Content to show if condition is true</div>
```

Conditions like ng-if else and ng-if else then are sub parts of ng-if .

EXAMPLE:

```
<h1>ngIf </h1>
 <p *ngIf="showMe">
  ShowMe is checked
</p>
<p *ngIf="!showMe">
  ShowMe is unchecked
</p>
 <h1>ngIf Else</h1>
 <p *ngIf="showMe; else elseBlock1">
  ShowMe is checked
</p>
```

OUTPUT:

ng-show and ng-hide Directives:

ng-show:

- Used to show an element based on a boolean expression.(i.e)Shows or hides the associated HTML element.

- The ng-Show directive in AngularJS is used to show or hide a given specific HTML element based on the expression provided to the ng-show attribute. In the background, the HTML element is shown or hidden by removing or adding the .ng-hide CSS class onto the element. It is a predefined CSS class which sets the display to none.

ng-hide:

- Used to hide an element based on a boolean expression.(or)
- Conditionally hides an HTML element based on the truthiness of an expression.
- The ng-hide directive in AngularJS is a function using which an element will be hidden if the expression is TRUE. If the Expression is FALSE, the element will be shown. In the background, the element is shown or hidden by removing or adding the .ng-hide CSS class onto the element

SYNTAX:

* Content to show if condition is true *

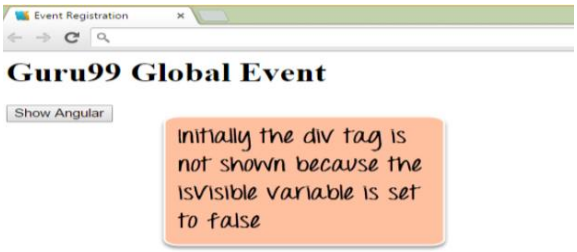
* Content to hide if condition is true *

Example: ng-show

```
<h1> Guru99 Global Event</h1>
<div ng-app="DemoApp" ng-controller="DemoController">
   <input type="button" value="Show Angular" ng-click="ShowHide()"/>
   <br><br><div ng-show = "IsVisible">Angular</div>
</div>
<script type="text/javascript">
   var app = angular.module('DemoApp',[]);
   app.controller('DemoController',function($scope){
      $scope.IsVisible = false;
```

```
    $scope.ShowHide = function(){
        $scope.IsVisible = true;
    }
    });
```

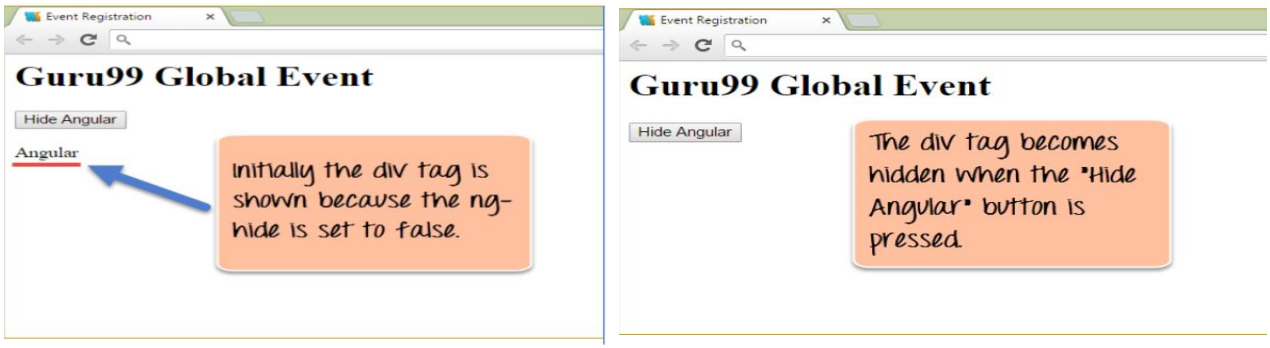
OUTPUT:



Example: ng-hide

```
<h1> Guru99 Global Event</h1>
<div ng-app="DemoApp" ng-controller="DemoController">
    <input type="button" value="Hide Angular" ng-click="ShowHide()"/>
    <br><br><div ng-hide="IsVisible">Angular</div>
</div>
<script type="text/javascript">
    var app = angular.module('DemoApp',[]);
    app.controller('DemoController',function($scope){
        $scope.IsVisible = false;
        $scope.ShowHide = function(){
            $scope.IsVisible = $scope.IsVisible = true;
        } });
</script>
```

OUTPUT:



ng-switch Directive:

- Used to conditionally render content based on multiple conditions.

ng-switch, provide a way to dynamically control the visibility and rendering of elements based on certain conditions. These directives enhance the flexibility and responsiveness of AngularJS applications.

- Angular evaluates the switch_expression at runtime and based on its value displays or removes the elements from the DOM.

SYNTAX:

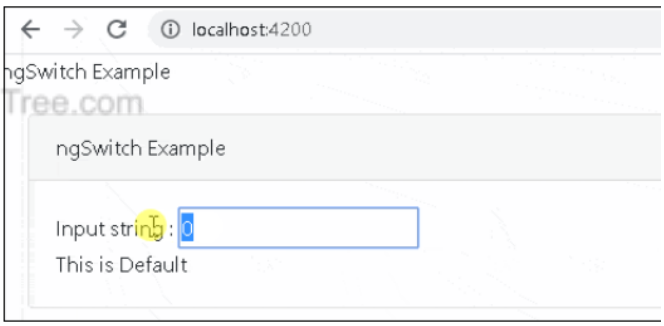
```
<div ng-switch="variable">
  <div ng-switch-when="value1">Content for value1</div>
  <div ng-switch-default>Default content</div>
</div>
```

EXAMPLE:

```
<div class='card'>
  <div class='card-header'>
    ngSwitch Example
  </div>
  <div class="card-body">
    Input string : <input type='text' [(ngModel)]="num" />
```

```
    <div [ngSwitch]="num">
      <div *ngSwitchCase="'1'">One</div>
      <div *ngSwitchCase="'2'">Two</div>
      <div *ngSwitchCase="'3'">Three</div>
      <div *ngSwitchCase="'4'">Four</div>
      <div *ngSwitchCase="'5'">Five</div>
      <div *ngSwitchDefault>This is Default</div>
    </div>
  </div>
</div>
```

OUTPUT:



STYLE DIRECTIVES

Style directives in AngularJS are a way of applying CSS styles to HTML elements dynamically, based on some conditions or expressions.

They can be useful for creating responsive and interactive web pages that change their appearance according to the user's actions or data changes.

TYPES:

There are two main types of style directives in AngularJS:

- ng-style and
- ng-class.
- Other types of Style directives in AngularJS includes:

- ng-blur
- ng-click and
- ng-disabled

PURPOSE:

The ng-style directive allows you to specify the style attribute for an HTML element as an object or an expression returning an object. The object consists of CSS properties and values, in key-value pairs.

ng-style directive:

The ng-style directive specifies the style attribute for the HTML element.

The value of the ng-style attribute must be an object, or an expression returning an object.

The object consists of CSS properties and values, in key value pairs.

The **ng-style Directive** in AngularJS is used to apply custom CSS styles on an HTML element. The expression inside the ng-style directive must be an object. It is supported by all the HTML elements.

SYNTAX:

<*element* ng-style="*expression*"></*element*>

expression: It specifies the particular style to be implemented for the respective element.

This is supported by all HTML elements.

Example program:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></sc
ript>
<body ng-app="myApp" ng-controller="myCtrl">
```

```
<h1 ng-style="myObj">Welcome</h1>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
  $scope.myObj = {
    "color" : "white",
    "background-color" : "coral",
    "font-size" : "60px",
    "padding" : "50px"
  }
});
</script>
</body>
</html>
```


OUTPUT:

In the above example,

We are using a HTML document format to implement AngularJS using <script> tag. Here, we declare the ng-style directive with a name “myObj”

that is defined in the script (AngularJS) tag to apply background color to the h1 element i.e, WELCOME.

ng-class directive:

- The ng-class directive dynamically binds one or more CSS classes to an HTML element.
- The value of the ng-class directive can be a string, an object, or an array.
- If it is a string, it should contain one or more, space-separated class names.
- As an object, it should contain key-value pairs, where the key is the class name of the class you want to add, and the value is a boolean value. The class will only be added if the value is set to true.
- As an array, it can be a combination of both. Each array element can be either a string, or an object, described as above.

SYNTAX:

<*element* ng-class="*expression*"></*element*>

Example program:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<style>
.sky {
   color:white;
   background-color:lightblue;
   padding:20px;
   font-family:"Courier New";
```

```
}
.tomato {
   background-color:coral;
   padding:40px;
   font-family:Verdana;
}
</style>
<body ng-app="">
<p>Choose a class:</p>
<select ng-model="home">
<option value="sky">Sky</option>
<option value="tomato">Tomato</option>
</select>
<div ng-class="home">
  <h1>Welcome Home!</h1>
  <p>I like it!</p>
</div>
</body>
</html>
```


OUTPUT:

Here we use the ng-class directive to choose between two options i.e., Sky and Tomato within the same class “Home”.

There are two types of ng-class directive. They are:

- ng-class-even
- ng-class-odd

ng-class-even:

The **ng-class-even Directive** in AngularJS is used to specify the CSS classes on every even appearance of HTML elements. It is used to dynamically bind classes on every even HTML element. If the expression inside the *ng-class-even directive* returns true then only the class is added else it is not added. The **ng-repeat directive** is required for the ng-class-even directive to work. It is supported by all HTML elements.

SYNTAX:

<element ng-class-even="expression">Contents...</element>

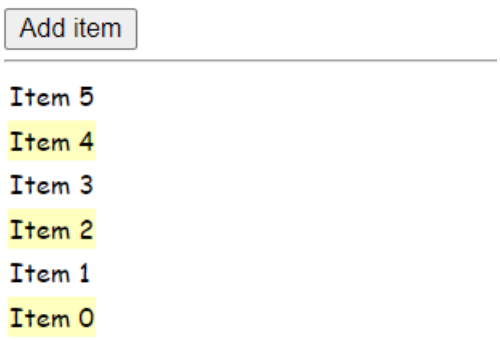
expression: This parameter returns more than one class name or simply specifies the CSS class for *even* appearance of HTML elements.

Example program:

```
<div ng-init="items=['Item 3', 'Item 2', 'Item 1', 'Item 0']">
  <button ng-click="items.unshift('Item ' + items.length)">Add item</button>
  <hr />
```

```
  <table>
    <tr ng-repeat="item in items" ng-class-even="'even'">
      <td>{{ item }}</td>
    </tr>
  </table>
</div>
```

OUTPUT:



Here the class “even” is used to highlight the even items everytime a new item is added by clicking “Add item” button.

ng-class-odd:

The **ng-class-odd Directive** in AngularJS is used to specify the CSS classes on every odd appearance of HTML elements. It is used to dynamically bind classes on every odd HTML element. If the expression inside the *ng-class-odd directive* returns true then only the class is added else it is not added. The **ng-repeat directive** is required for the ng-class-odd directive to work. This directive can be utilized for the styling of the items in a list or rows in a table, although, can be used for remaining HTML elements, as well. It is supported by all HTML elements.

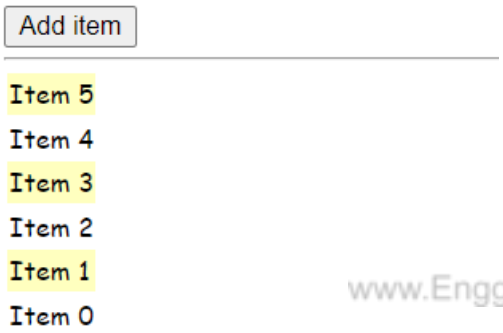
SYNTAX:

<element ng-class-odd="expression">Contents... </element>

Example program:

```
<div ng-init="items=['Item 3', 'Item 2', 'Item 1', 'Item 0']">
  <button ng-click="items.unshift('Item ' + items.length)">Add item</button>
  <hr />
  <table>
    <tr ng-repeat="item in items" ng-class-odd="'odd'">
      <td>{{ item }}</td>
    </tr>
  </table>
</div>
```

OUTPUT:



Here the class “odd” is used to highlight the odd items everytime a new item is added by clicking “Add item” button.

ng-click directive:

- The **ng-click Directive** in AngluarJS is used to apply custom behavior when an element is clicked. It can be used to show/hide some element or it can pop up an alert when the button is clicked.
- In other words, The ng-click directive tells AngularJS what to do when an HTML element is clicked.

SYNTAX:

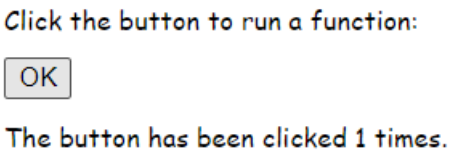
<*element* ng-click="*expression*"></*element*>

expression: It specifies when the particular element is clicked then the specific expression will be evaluated.

Example program:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<body ng-app="myApp">
<div ng-controller="myCtrl">
  <p>Click the button to run a function:</p>
  <button ng-click="myFunc()">OK</button>
  <p>The button has been clicked {{count}} times.</p>
</div>
<script>
angular.module('myApp', [])
  .controller('myCtrl', ['$scope', function($scope) {
    $scope.count = 0;
    $scope.myFunc = function() {
      $scope.count++;
    };
  }]);
</script>
</body>
</html>
```

OUTPUT:



ng-blur directive:

- The **ng-blur Directive** in AngularJS is fired when an HTML element loses their focus.
- It doesn't override with element's original onblur event i.e. both the ng-blur expression and original onblur event will execute.

SYNTAX:

<element ng-blur="expression">Contents... </element>

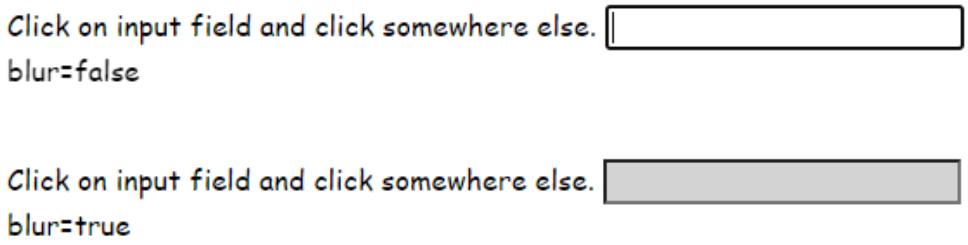
expression: It refers to the variable or the expression to be evaluated.

Note: The ng-blur directive is supported by <input>, <a>, <select> and <textarea>.

Example program:

```
<style>
.blurClass {background-color: lightgray;}
</style>

<div ng-app="myApp" ng-controller='MyCtrl' ng-init="blur=false;">
  Click on input field and click somewhere else.
  <input type="text"  ng-class="{blurClass:blur}"  ng-blur="blur=true;"
          ng-blur= "blur=false;"><br>
  blur={{blur}}
</div>
```

```
<script
src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.3/angular.min.js"></script>
<script>
  var module = angular.module('myApp', []);
  module.controller('MyCtrl', function($scope) {});
</script>
```



OUTPUT

ANGULARJS CONTROLLERS

- AngularJS controllers **control the data** of AngularJS applications.
- AngularJS controllers are regular **JavaScript Objects**.
- AngularJS applications are controlled by controllers.
- The **ng-controller** directive defines the application controller.
- A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

Controller Methods:

A controller can have methods (variables as functions)

The example demonstrated a controller object with two properties: lastName and firstName.

EXAMPLE:

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

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<body>
<div ng-app="myApp" ng-controller="personCtrl">
First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{fullName()}}
</div>
<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
    $scope.firstName = "John";
    $scope.lastName = "Doe";
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    };
});
</script>
</body>
</html>
```


OUTPUT:

First Name: John

Last Name: Doe

Full Name: John Doe

Controllers In External Files:

In larger applications, it is common to store controllers in external files.

Just copy the code between the <script> tags into an external file named personController.js:

EXAMPLE:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<body>
<div ng-app="myApp" ng-controller="personCtrl">
First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{fullName()}}
</div>
<script src="personController.js"></script>
</body>
</html>
```


personController.js

```
<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
    $scope.firstName = "John";
    $scope.lastName = "Doe";
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
```

```
    };
});
</script>
```

OUTPUT:

First Name: John

Last Name: Doe

AngularJS Scope:

- The scope is the binding part between the HTML (view) and the JavaScript (controller).
- The scope is an object with the available properties and methods.
- The scope is available for both the view and the controller.
- When you make a controller in AngularJS, you pass the $scope object as an argument

Root Scope:

- All applications have a $rootScope which is the scope created on the HTML element that contains the ng-app directive.
- The rootScope is available in the entire application.

ANGULARJS FILTERS

- Filters can be added in AngularJS to format data.
- AngularJS provides filters to transform data:

- currency Format a number to a currency format.
- date Format a date to a specified format.
- filter Select a subset of items from an array.
- json Format an object to a JSON string.

- limitTo Limits an array/string, into a specified number of elements/characters.
- lowercase Format a string to lower case.
- number Format a number to a string.
- orderBy Orders an array by an expression.
- uppercase Format a string to upper case.

Adding Filters to Expressions

- Filters can be added to expressions by using the pipe character |, followed by a filter.

UPPERCASE:

The uppercase filter format strings to upper case:

EXAMPLE:

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



```
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<body>
<div ng-app="myApp" ng-controller="personCtrl">
<p>The name is {{ lastName | uppercase }}</p>
</div>
<script>
angular.module('myApp', []).controller('personCtrl', function($scope) {
   $scope.firstName = "John",
   $scope.lastName = "Doe"
});
</script>
</body>
</html>
```

OUTPUT:

The name is DOE

LOWERCASE:

->The lowercase filter format strings to lower case:

-> From the above example, instead of uppercase, you put lowercase.(inside the tag)

EXAMPLE:

```
<p>The name is {{ lastName | lowercase }}</p>
```

Adding Filters to Directives

Filters are added to directives, like ng-repeat, by using the pipe character |, followed by a filter.

ORDERBY:

The orderBy filter sorts an array:

EXAMPLE:

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<body>
<div ng-app="myApp" ng-controller="namesCtrl">
<p>Looping with objects:</p>
<ul>
  <li ng-repeat="x in names | orderBy:'country'">
    {{ x.name + ', ' + x.country }}
  </li>
```

```
</ul>
</div>
<script>
angular.module('myApp', []).controller('namesCtrl', function($scope) {
  $scope.names = [
    {name:'Jani',country:'Norway'},
    {name:'Carl',country:'Sweden'},
    {name:'Margareth',country:'England'},
    {name:'Hege',country:'Norway'},
    {name:'Joe',country:'Denmark'},
    {name:'Gustav',country:'Sweden'},
    {name:'Birgit',country:'Denmark'},
    {name:'Mary',country:'England'},
    {name:'Kai',country:'Norway'}
    ];
});
</script>
</body>
```


OUTPUT:

Looping with objects:

- Joe, Denmark
- Birgit, Denmark
- Margareth, England
- Mary, England
- Jani, Norway
- Hege, Norway
- Kai, Norway
- Carl, Sweden
- Gustav, Sweden

CURRENCY:

The currency filter formats a number as currency:

EXAMPLE:

```
<div ng-app="myApp" ng-controller="costCtrl">
<h1>Price: {{ price | currency }}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('costCtrl', function($scope) {
   $scope.price = 58;
});
</script>
<p>The currency filter formats a number to a currency format.</p>
```

OUTPUT:

Price: $58.00

The currency filter formats a number to a currency format.

FILTER:

- The filter filter selects a subset of an array.
- The filter filter can only be used on arrays, and it returns an array containing only the matching items.

EXAMPLE

Return the names that contains the letter "i":

```
<div ng-app="myApp" ng-controller="namesCtrl">
<ul>
  <li ng-repeat="x in names | filter : 'i'">
    {{ x }}
  </li>
```

```
</ul>
</div>
```

Sort an Array Based on User Input:

The code for sorting an array

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<body>
<p>Click the table headers to change the sorting order:</p>
<div ng-app="myApp" ng-controller="namesCtrl">
<table border="1" width="100%">
<tr>
<th ng-click="orderByMe('name')">Name</th>
<th ng-click="orderByMe('country')">Country</th>
</tr>
<tr ng-repeat="x in names | orderBy:myOrderBy">
<td>{{x.name}}</td>
<td>{{x.country}}</td>
</tr>
</table>
</div>
<script>
angular.module('myApp', []).controller('namesCtrl', function($scope) {
  $scope.names = [
    {name:'Jani',country:'Norway'},
    {name:'Carl',country:'Sweden'},
    {name:'Margareth',country:'England'},
```

```
        {name:'Hege',country:'Norway'},
        {name:'Joe',country:'Denmark'},
        {name:'Gustav',country:'Sweden'},
        {name:'Birgit',country:'Denmark'},
        {name:'Mary',country:'England'},
        {name:'Kai',country:'Norway'}
        ];
    $scope.orderByMe = function(x) {
        $scope.myOrderBy = x;
    }
});
</script>
</body>
</html>
```

OUTPUT:

Click the table headers to change the sorting order:

Name	Country
Jani	Norway
Carl	Sweden
Margareth	England
Hege	Norway
Joe	Denmark
Gustav	Sweden
Birgit	Denmark
Mary	England
Kai	Norway

After clicking the header:

Click the table headers to change the sorting order:

Name	Country
Birgit	Denmark
Carl	Sweden
Gustav	Sweden
Hege	Norway
Jani	Norway
Joe	Denmark
Kai	Norway
Margareth	England
Mary	England

ANGULARJS ROUTING

The ng-Route module helps your application to become a Single Page Application.

What is Routing in AngularJS?

- If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the ng-Route module.
- The ng-Route module *routes* your application to different pages without reloading the entire application.

EXAMPLE:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-
route.js"></script>
<body ng-app="myApp">
<p><a href="#/!">Main</a></p>
```

```
<a href="#!red">Red</a>
<a href="#!green">Green</a>
<a href="#!blue">Blue</a>
<div ng-view></div>
<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
   $routeProvider
   .when("/", {
      templateUrl : "main.htm"
   })
   .when("/red", {
      templateUrl : "red.htm"
   })
   .when("/green", {
      templateUrl : "green.htm"
   })
   .when("/blue", {
      templateUrl : "blue.htm"
   });
});
</script>
<p>Click on the links to navigate to "red.htm", "green.htm", "blue.htm", or back
to "main.htm"</p>
</body>
</html>
```


OUTPUT:

Main

Red

Green

Blue

Main

Click on the links to navigate to "red.htm", "green.htm", "blue.htm", or back to "main.htm"

What does we Need?

- To make the applications ready for routing, we must include the AngularJS Route module:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-route.js"></script>
```

- Then we must add the ngRoute as a dependency in the application module:

```
var app = angular.module("myApp", ["ngRoute"]);
```

- Now the application has access to the route module, which provides the $routeProvider. Use the $routeProvider to configure different routes in the application:

```
app.config(function($routeProvider) {
  $routeProvider
  .when("/", {
    templateUrl : "main.htm"
  })
  .when("/red", {
    templateUrl : "red.htm"
  })
  .when("/green", {
    templateUrl : "green.htm"
  })
  .when("/blue", {
    templateUrl : "blue.htm"
```

```
  });
});
```

Where Does it Go?

- The application needs a container to put the content provided by the routing.
- This container is the ng-view directive.
- There are three different ways to include the ng-view directive in the application:

EXAMPLE:

☺<div ng-view>

☺<ng-view></ng-view>

☺

Applications can only have one ng-view directive, and this will be the placeholder for all views provided by the route.

$routeProvider

With the $routeProvider you can define what page to display when a user clicks a link.

EXAMPLE:

Define a $routerProvider:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-
route.js"></script>
<body ng-app="myApp">
```

```
<p><a href="#/!">Main</a></p>
<a href="#!london">City 1</a>
<a href="#!paris">City 2</a>
<p>Click on the links to read about London and Paris.</p>
<div ng-view></div>
<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
   $routeProvider
   .when("/", {
      templateUrl : "main.htm"
   })
   .when("/london", {
      templateUrl : "london.htm"
   })
   .when("/paris", {
      templateUrl : "paris.htm"
   });
});
</script>
</body>
</html>
```


OUTPUT:

Main

City 1

City 2

Click on the links to read about London and Paris.

Main

->Define the $routeProvider using the config method of your application. Work registered in the config method will be performed when the application is loading.

Controllers

With the $routeProvider you can also define a controller for each "view".

EXAMPLE:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-
route.js"></script>
<body ng-app="myApp">
<p><a href="#/!">Main</a></p>
<a href="#!london">City 1</a>
<a href="#!paris">City 2</a>
<p>Click on the links.</p>
<p>Note that each "view" has its own controller which each gives the "msg"
variable a value.</p>
<div ng-view></div>
<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
  $routeProvider
  .when("/", {
    templateUrl : "main.htm",
  })
```

```
    .when("/london", {
        templateUrl : "london.htm",
        controller : "londonCtrl"
    })
    .when("/paris", {
        templateUrl : "paris.htm",
        controller : "parisCtrl"
    });
});
app.controller("londonCtrl", function ($scope) {
    $scope.msg = "I love London";
});
app.controller("parisCtrl", function ($scope) {
    $scope.msg = "I love Paris";
});
</script>
</body>
</html>
```


OUTPUT:

Main

City 1

City 2

Click on the links.

Note that each "view" has its own controller which each gives the "msg" variable a value.

Main

The otherwise method

- In the previous examples we have used the when method of the $routeProvider.
- We can also use the otherwise method, which is the default route when none of the others get a match.

EXAMPLE:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-
route.js"></script>
<body ng-app="myApp">
<p><a href="#/!">Main</a></p>
<a href="#!banana">Banana</a>
<a href="#!tomato">Tomato</a>
<p>Click on the links to change the content.</p>
<p>Use the "otherwise" method to define what to display when none of the
links are clicked.</p>
<div ng-view></div>
<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
  $routeProvider
  .when("/banana", {
    template : "<h1>Banana</h1><p>Bananas contain around 75%
water.</p>"
  })
  .when("/tomato", {
```

```
    template : "<h1>Tomato</h1><p>Tomatoes contain around 95%
water.</p>"
  })
  .otherwise({
    template : "<h1>Nothing</h1><p>Nothing has been selected</p>"
  });
});
</script>
</body>
</html>
```

OUTPUT:

Main

Banana

Tomato

Click on the links to change the content.

Use the "otherwise" method to define what to display when none of the links are clicked.

Nothing has been selected

MODULES

- An AngularJS module defines an application.
- The module is a container for the different parts of an application.
- The module is a container for the application controllers.
- Controllers always belong to a module.

Creating a Module:

- A module is created by using the AngularJS function angular.module
- The "myApp" parameter refers to an HTML element in which the application will run.

- Now you can add controllers, directives, filters, and more, to your AngularJS application.

EXAMPLE:

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
   $scope.firstName = "John";
   $scope.lastName = "Doe";
});
</script>
</body>
</html>
```

OUTPUT:

John Doe

ANGULAR FORMS

Forms are collection of controls that is input field , buttons , checkbox and these can be validated real time . Forms in AngularJS provides data-binding and validation of input controls. As soon as a user of the form completes writing a

field and moves to the next one it get validated and suggests the user where he might have gone wrong .

So a form can be consisting of many number of controls :

1. Input elements (Input field)
2. Select elements (Checkbox , Radiobox)
3. Button elements (button)
4. Textarea elements
5. SelectBox (Dropdowns)

Data-Binding

Input field :

Input controls provides data-binding by using the ng-model directive.

SYNTAX :

<input type="text" ng-model="firstname">

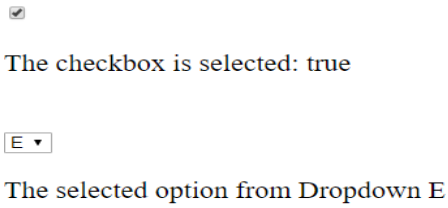
Checkbox and Selectbox (Dropdowns) :

In a form , the variable defined in ngModel is stored with truie on getting selected otherwise false . in Select , the selected value gets stored in the variable defined in ngModel .

EXAMPLE :

```
<form>
    <input id="myVar" type="checkbox" ngModel name="myVar" #myVar="ngModel">
    <p>The checkbox is selected: {{myVar.value}}</p>
<br />
    <select ngModel name="mychoice" #myChoice="ngModel">
```

```
            <option>A</option>
            <option>E</option>
            <option>I</option>
            <option>O</option>
            <option>U</option>
        </select>
        <p>The selected option from Dropdown {{ myChoice.value }}</p>
</form>
```



OUTPUT :

Radiobuttons and Buttons :

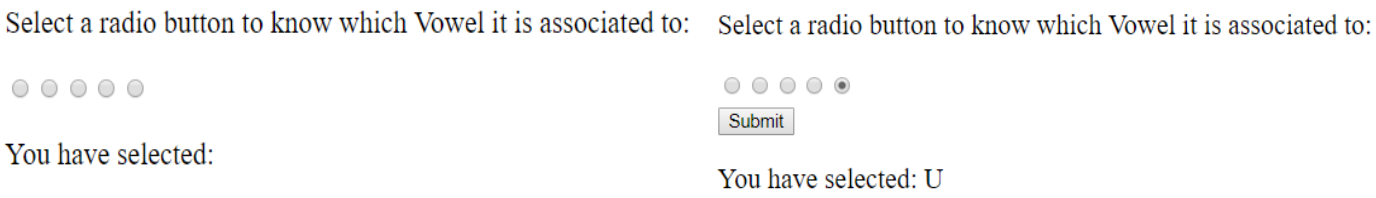
Radio buttons used in the forms should allow only one field to be selected at a time to make sure this is the case we should associate it with only ngModel .

EXAMPLE :

```
<form>
        <p>Select a radio button to know which Vowel it is associated to:</p>
        <input value="A" type="radio" ngModel name="myVar"
#myVar="ngModel">
        <input value="E" type="radio" ngModel name="myVar"
#myVar="ngModel">
        <input value="I" type="radio" ngModel name="myVar"
#myVar="ngModel">
```

```
        <input value="O" type="radio" ngModel name="myVar"
#myVar="ngModel">
        <input value="U" type="radio" ngModel name="myVar"
#myVar="ngModel">
<br/><button *ngIf='myVar.touched'>Submit</button>
</form>
<p>You have selected: {{myVar.value}}</p>
```

OUTPUT :



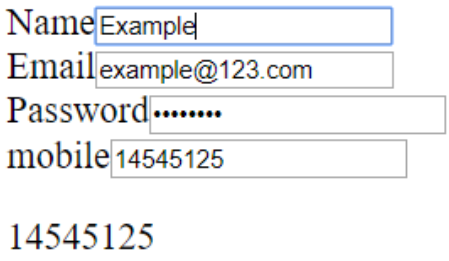
Before selecting **After selecting**

EXAMPLE FOR CREATING A FORM :

```
<form>
        <div class="form-group">
            <label for="firstName">Name</label>
            <input type="text"
            id="firstName"
            placeholder="Name">
    </div>
        <div class="form-group">
            <label for="email">Email</label>
            <input
            type="text"
```

```
            id="email"
            placeholder="Email">
        </div>
        <div class="form-group">
            <label for="password">Password</label>
            <input
                type="password"
                id="password"
                placeholder="Password">
        </div>
        <div class="form-group">
            <label for="phone">mobile</label>
            <input
            type="text"
            id="phone"
            ngModel name="phone"
            #phone="ngModel"
            placeholder="Mobile">
        </div>
</form>
<p>{{ phone.value }}</p>
```

OUTPUT :



FORM VALIDATION :

AngularJS offers client-side form validation.AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state. AngularJS also holds information about whether they have been touched, or modified, or not. You can use standard HTML5 attributes to validate input, or you can make your own validation functions.

Form input fields have the following states:

- **$untouched:** It shows that field has not been touched yet.
- **$touched:** It shows that field has been touched.
- **$pristine:** It represents that the field has not been modified yet.
- **$dirty:** It illustrates that the field has been modified.
- **$invalid:** It specifies that the field content is not valid.
- **$valid:** It specifies that the field content is valid.

These all are the properties of the **input field** which can be either true or false.

Forms have the following states:

- **$pristine:** It represents that the fields have not been modified yet.
- **$dirty:** It illustrates that one or more fields have been modified.
- **$invalid:** It specifies that the form content is not valid.
- **$valid:** It specifies that the form content is valid.
- **$submitted:** It specifies that the form is submitted.

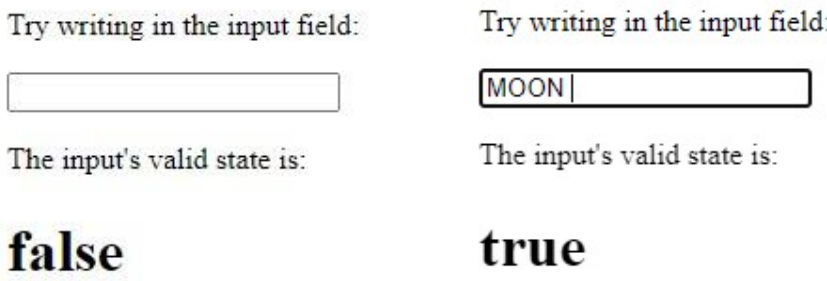
These all are the properties of the **form** which can be either true or false. These states can be used to show meaningful messages to the user.

EXAMPLE :

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

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></scr
ipt>
<body ng-app="">
<p>Try writing in the input field:</p>
<form name="myForm">
<input name="myInput" ng-model="myInput" required>
</form>
<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
</body>
</html>
```

OUTPUT :



BEFORE WRITING **AFTER WRITING**

WEB SERVICES

DEFINITION:

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents. A web service is a

collection of open protocols and standards used for exchanging data between applications or systems.

The Internet is the worldwide connectivity of hundreds of thousands of computers of various types that belong to multiple networks. On the World Wide Web, a web service is a standardized method for propagating messages between client and server applications. A web service is a software module that is intended to carry out a specific set of functions. Web services in cloud computing can be found and invoked over the network.

The web service would be able to deliver functionality to the client that invoked the web service.

A web service is a set of open protocols and standards that allow data to be exchanged between different applications or systems. Web services can be used by software programs written in a variety of programming languages and running on a variety of platforms to exchange data via computer networks such as the Internet in a similar way to inter-process communication on a single computer.

Any software, application, or cloud technology that uses standardized web protocols (HTTP or HTTPS) to connect, interoperate, and exchange data messages – commonly XML (Extensible Markup Language) – across the internet is considered a web service.

Web services have the advantage of allowing programs developed in different languages to connect with one another by exchanging data over a web service between clients and servers. A client invokes a web service by submitting an XML request, which the service responds with an XML response.

Functions of Web Services

- It's possible to access it via the internet or intranet networks.
- XML messaging protocol that is standardized.
- Operating system or programming language independent.

- Using the XML standard, it is self-describing.
- A simple location approach can be used to locate it.

Components of Web Service

XML and HTTP is the most fundamental web services platform. The following components are used by all typical web services:

SOAP (Simple Object Access Protocol)

SOAP stands for "Simple Object Access Protocol." It is a transport-independent messaging protocol. SOAP is built on sending XML data in the form of SOAP Messages. A document known as an XML document is attached to each message. Only the structure of the XML document, not the content, follows a pattern. The best thing about Web services and SOAP is that everything is sent through HTTP, the standard web protocol.

A root element known as the element is required in every SOAP document. In an XML document, the root element is the first element. The "envelope" is separated into two halves. The header comes first, followed by the body. The routing data, or information that directs the XML document to which client it should be sent to, is contained in the header. The real message will be in the body.

UDDI (Universal Description, Discovery, and Integration)

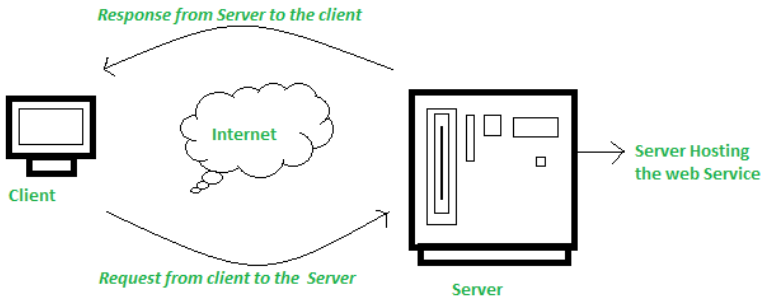
UDDI is a standard for specifying, publishing and discovering a service provider's online services. It provides a specification that aids in the hosting of data via web services. UDDI provides a repository where WSDL files can be hosted so that a client application can discover a WSDL file to learn about the various actions that a web service offers. As a result, the client application will have full access to the UDDI, which serves as a database for all WSDL files. The UDDI registry will hold the required information for the online service,

just like a telephone directory has the name, address, and phone number of a certain individual. So that a client application may figure out where it is.

WSDL (Web Services Description Language)

If a web service can't be found, it can't be used. The client invoking the web service should be aware of the location of the web service. Second, the client application must understand what the web service does in order to invoke the correct web service. The WSDL, or Web services description language, is used to accomplish this. The WSDL file is another XML-based file that explains what the web service does to the client application. The client application will be able to understand where the web service is located and how to use it by using the WSDL document.

How Does Web Service Work?



The diagram depicts a very simplified version of how a web service would function. The client would use requests to send a sequence of web service calls to a server that would host the actual web service.

Remote procedure calls are what are used to make these requests. Calls to methods hosted by the relevant web service are known as Remote Procedure Calls (RPC). Example: Flipkart offers a web service that displays prices for items offered on Flipkart.com. The front end or presentation layer can be

written in .Net or Java, but the web service can be communicated using either programming language. The data that is exchanged between the client and the server, which is XML, is the most important part of a web service design. XML (Extensible markup language) is a simple intermediate language that is understood by various programming languages. It is a counterpart to HTML. As a result, when programs communicate with one another, they do so using XML. This creates a common platform for applications written in different programming languages to communicate with one another.
For transmitting XML data between applications, web services employ SOAP (Simple Object Access Protocol). The data is sent using standard HTTP. A SOAP message is data that is sent from the web service to the application. An XML document is all that is contained in a SOAP message. The client application that calls the web service can be created in any programming language because the content is written in XML.

Features/Characteristics Of Web Service

Web services have the following features:

(a) XML Based: The information representation and record transportation layers of a web service employ XML. There is no need for networking, operating system, or platform binding when using XML. At the middle level, web offering-based applications are highly interoperable.

(b) Loosely Coupled: A customer of an internet service provider isn't necessarily directly linked to that service provider. The user interface for a web service provider can change over time without impacting the user's ability to interact with the service provider. A strongly coupled system means that the patron's and server's decisions are inextricably linked, indicating that if one interface changes, the other should be updated as well.

A loosely connected architecture makes software systems more manageable and allows for easier integration between different structures.

(c) Capability to be Synchronous or Asynchronous: Synchronicity refers to the client's connection to the function's execution. The client is blocked and the client has to wait for the service to complete its operation, before continuing in synchronous invocations. Asynchronous operations allow a client to invoke a task and then continue with other tasks. Asynchronous clients get their results later, but synchronous clients get their effect immediately when the service is completed. The ability to enable loosely linked systems requires asynchronous capabilities.

(d) Coarse-Grained: Object-oriented systems, such as Java, make their services available through individual methods. At the corporate level, a character technique is far too fine an operation to be useful. Building a Java application from the ground, necessitates the development of several fine-grained strategies, which are then combined into a rough-grained provider that is consumed by either a buyer or a service. Corporations should be coarse-grained, as should the interfaces they expose. Web services generation is an easy approach to define coarse-grained services that have access to enough commercial enterprise logic.

(e) Supports Remote Procedural Call: Consumers can use an XML-based protocol to call procedures, functions, and methods on remote objects utilizing web services. A web service must support the input and output framework exposed by remote systems.

Enterprise-wide component development Over the last few years, JavaBeans (EJBs) and.NET Components have become more prevalent in architectural and enterprise deployments. A number of RPC techniques are used to allocate and

access both technologies.

A web function can support RPC by offering its own services, similar to those of a traditional role, or by translating incoming invocations into an EJB or.NET component invocation.

(f) Supports Document Exchanges: One of XML's most appealing features is its simple approach to communicating with data and complex entities. These records can be as simple as talking to a current address or as complex as talking to an entire book or a Request for Quotation. Web administrations facilitate the simple exchange of archives, which aids incorporate reconciliation. The web benefit design can be seen in two ways:

(i) The first step is to examine each web benefit on-screen character in detail.

(ii) The second is to take a look at the rapidly growing web benefit convention stack.

Advantages Of Web Service

Using web services has the following advantages:

(a) Business Functions can be exposed over the Internet: A web service is a controlled code component that delivers functionality to client applications or end-users. This capability can be accessed over the HTTP protocol, which means it can be accessed from anywhere on the internet. Because all apps are now accessible via the internet, Web services have become increasingly valuable. Because all apps are now accessible via the internet, Web services have become increasingly valuable. That is to say, the web service can be located anywhere on the internet and provide the required functionality.

(b) Interoperability: Web administrations allow diverse apps to communicate with one another and exchange information and services. Different apps can

also make use of web services. A .NET application, for example, can communicate with Java web administrations and vice versa. To make the application stage and innovation self-contained, web administrations are used.

(c) Communication with Low Cost: Because web services employ the SOAP over HTTP protocol, you can use your existing low-cost internet connection to implement them. Web services can be developed using additional dependable transport protocols, such as FTP, in addition to SOAP over HTTP.

(d) A Standard Protocol that Everyone Understands: Web services communicate via a defined industry protocol. In the web services protocol stack, all four layers (Service Transport, XML Messaging, Service Description, and Service Discovery) use well-defined protocols.

(e) Reusability: A single web service can be used simultaneously by several client applications.

WEB APPLICATION

DEFINITION:

A web-application is an application program that is usually stored on a remote server, and users can access it through the use of **Software** known as **web-browser.**

In general, a web application can contain online shops (or we can also say them e-commerce shops), webmail's, calculators, social media platforms, etc. There is also some kind of web application that usually requires a special kind of web browser to access them. We cannot access those kinds of web applications by using regular web- browsers. However, most of the web applications available on the internet can be accessed using a **standard web browser**.

If we talk about the web application in general, a web application usually uses a combination of the server-side scripts such as **PHP, ASP,** for handling the information/ data storage and retrieval of the data.

Some of them also use the client-side scripts such as **JavaScript, HTML** to represent the data/information in front of the users, and some of the web applications are also using both **server-side** and **client-side** at the same time.

It allows the users to communicate with the organization or companies by using the online form, online forums, shopping carts, content management system, and much more.

Apart from that web applications also allow its users to create documents, share them, or share the data/ information. By using the web application, users can collaborate on same projects by event when they are not available on the same geographical location.

After knowing that what a web application is, there may be a question hitting in mind that how it will work.

How does a web- application work?

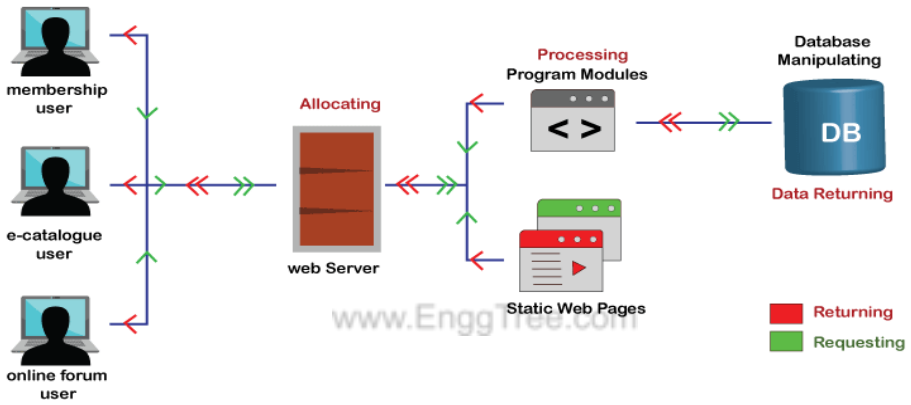
In general, web-application does not require downloading them because, as we already discussed, the web application is a computer program that usually resides on the remote server. Any user can access it by using one of the standard web browsers such as **Google Chrome, Safari, Microsoft Edge, etc.,** and most of them are available free for everyone.

A web application are generally coded using the languages supported by almost every web-browsers such as HTML, JavaScript because these are the languages that rely on the web browsers to render the program executable.

Some of the web applications are entirely static due to which they not required any processing on the server at all while, on the other hand, some web applications are dynamic and require server-side processing.

To operate a web- application, we usually required a web server (or we can say some space on the web-server for our programs/application's code) to manage the clients' upcoming requests and required an application server.

The application server performs the task that requested by the clients, which also may need a database to store the information sometimes. Application server

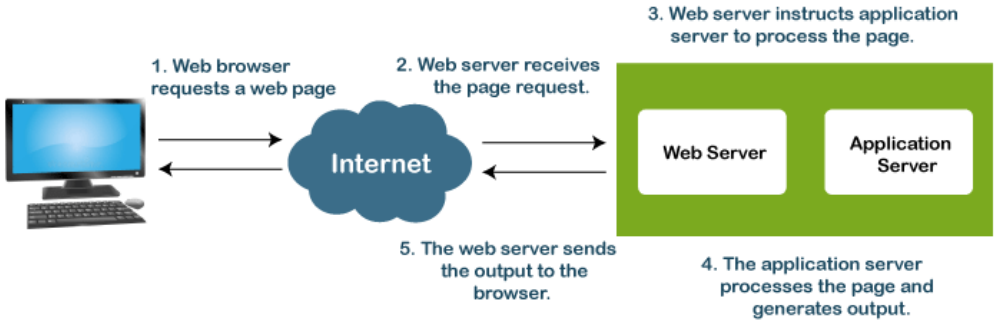


technologies range from **ASP.NET, ASP**, and **ColdFusion to PHP and JSP**.

A standard web application usually has short development cycles and can be easily developed with a small team of developers. As we all know, most of the currently available web applications on the internet are written using the programming languages such as the **HTML (or HyperText Markup Language), CSS(or Cascading Style Sheets), and Javascript** that are used in creating **front-end interface (Client-side programming).**

To create the web applications script, server-side programming is done by using programming languages such as **Java, Python, PHP, and Ruby,** etc. **Python** and **Java** are the languages that are usually used for server-side programming.

The Flow of the Web Application



1. In general, a user sends a request to the web-server using web browsers such as **Google Chrome, Microsoft Edge, Firefox**, etc over the **internet**.
2. Then, the request is forwarded to the appropriate web **application server** by the **web-server**.
3. Web application server performs the requested operations/ tasks like **processing the database, querying the databases; produces** the result of the requested data.
4. The obtained result is sent to the web-server by the web application server along with the requested data/information or processed data.
5. The web server responds to the user with the requested or processed data/information and provides the result to the user's screen .

Benefits of a web application

Let see some of the significant benefits offered by a web application:

- Any typical web application can run or accessible on any operating system such as the Windows, Mac, Linux as long as the browser is compatible.
- A web application is usually not required to install in the hard drive of the computer system, thus it eliminates all the issues related to the space limitation.

- All the users are able to access the same version of the web application, which eliminates all compatibility issues.
- It also reduces software piracy in subscription-based web applications, for example, **SAAS (or Software as a service).**
- They also reduce the expense for end-users, business owners because the maintenance needed by the business is significantly less.
- Web applications are flexible. A user can work from any geographical location as long as he has a working internet connection.
- It just takes a moment to create a new user by providing a username, password, and URL, and it's all.
- After the availability of the cloud, storage space is now virtually unlimited as long as you can afford it.
- A web application can be programmed to run on a wide variety of operating systems, unlike native applications that can run on a particular platform.

- Any standard web application is developed with some basic programming languages like HTML, CSS that are compatible and well known among the IT professionals.

Disadvantages of the Web Applications

As we all know, there are two sides of anything; if something has some advantages, it may also have limitations/ disadvantages. Consider the following disadvantages of the web applications.

- Internet connection is necessary to access any web application, and without an internet connection, anyone can't use any of the web applications. It is very typical to get an internet connection in our modern cities, still rural area internet connectivity not so well.

- Several people in business believe that their data on the cloud environment is no that secure and likes to stick with old methods; they even don't want to use new methods.
- As we all know that many users like to use different web browsers according to their needs and choices. So while creating a web application, you must remember that your application must support several web browsers, including new and old versions of browsers.
- Speed-related issues are also affecting the web application's performance because there are several factors on which the performance of a web application depends, and these all factors affect the performance of the web application in their own way.
- If a user's web application faces any kind of issues, or if he does not have a good quality corporate website, his web application will not be going to run correctly, smoothly.
- A user must have to spend enough money to maintain the good condition of his web application, provide an update whenever an issue occurs, and make an attractive user interface, which is not so cheap at all.
- A web application must be programmed/ coded in such a way that it will be run regardless of the device's operating system.
- A web application may face some issues while running on Windows, Android, or several other operating systems if it is not responsive.

There are several advantages and disadvantages of web applications; it is impossible to discuss them all at once. So in the above, we only discussed some critical and useful points that may help you quickly understand the pros and cons of the web application.

FRAMEFORKS

DEFINITION:

Web Application Framework or simply "web framework" is a software framework that is designed to support the development of web applications including web services, web resources, and web APIs. Frameworks are, in short, libraries that help you develop your application faster and smarter!

Nowadays, the number of Web Frameworks has increased greatly. To help you pick up the most suitable one for your Web Application, we have compiled a list of 10 best frameworks available online, in your preferred language.

1. Ruby on Rails

Ruby on Rails is an extremely productive web application framework written by David Heinemeier Hansson. One can develop an application at least ten times faster with Rails than a typical Java framework. Moreover, Rails includes everything needed to create a database-driven web application, using the Model-View-Controller pattern.

Websites using Ruby on Rails are GroupOn, UrbanDictionary, AirBnb, Shopify, Github

2. Django

Django is another framework that helps in building quality web applications. It was invented to meet fast-moving newsroom deadlines while satisfying the tough requirements of *experienced Web developers*. Django developers say the applications are ridiculously fast, secure, scalable, and versatile.

Websites using Django are Disqus, Pinterest, Instagram, Quora, etc.

3. Angular(Also, know as Angular JS)

Angular is a framework by Google (originally developed by Misko Hevery and Adam Abrons) which helps us in building powerful Web Apps. It is a

framework to build large scale and high-performance web applications while keeping them as easy-to-maintain. There are a huge number of web apps that are built with Angular.

Websites using Angular are Youtube on PS3, Weather, Netflix, etc.

4. ASP.NET

ASP.NET is a framework developed by Microsoft, which helps us to build robust web applications for PC, as well as mobile devices. It is a high performance and lightweight framework for building Web Applications using .NET. All in all, a framework with Power, Productivity, and Speed.

Websites using ASP.NET are GettyImages, TacoBell, StackOverflow, etc.

5. METEOR

Meteor or MeteorJS is another framework that gives one a radically simpler way to build realtime mobile and web apps. It allows for rapid prototyping and produces cross-platform (Web, Android, iOS) code. Its cloud platform, Galaxy, greatly simplifies deployment, scaling, and monitoring.

Websites using Meteor are HaggleMate, WishPool, Telescope, etc.

6. Laravel

Laravel is a framework created by Taylor Otwell in 2011 and like all other modern frameworks, it also follows the MVC architectural pattern. Laravel values Elegance, Simplicity, and Readability. One can right away start learning and developing Laravel with Laracasts which has hundreds of tutorials in it.

Websites using Laravel are Deltanet Travel, Neighbourhood Lender, etc.

7. Express

Express or Expressjs is a minimal and flexible framework that provides a robust set of features for web and mobile applications. It is relatively minimal meaning many features are available as plugins. Express facilitates the rapid development of Node.js based Web applications. Express is also one major component of the MEAN software bundle.

Websites using Express are Storify, Myspace, LearnBoost, etc.

8. Spring

Spring, developed by Pivotal Software, is the most popular application development framework for enterprise Java. Myriads of developers around the globe use Spring to create high performance and robust Web apps. Spring helps in creating simple, portable, fast, and flexible JVM-based systems and applications.

Websites using spring are Mascus, Allocine, etc.

9. Play

Play is one of the modern web application framework written in Java and Scala. It follows the MVC architecture and aims to optimize developer productivity by using convention over configuration, hot code reloading, and display of errors in the browser. Play quotes itself as “The High-Velocity Web Framework”.

Websites using PLAY are LinkedIn, Coursera, LendUp, etc.

10. CodeIgniter

CodeIgniter, developed by EllisLab, is a famous web application framework to build dynamic websites. It is loosely based on MVC architecture since Controller classes are necessary but models and views are optional. CodeIgnitor promises with exceptional performance, nearly zero-configuration, and no large-scale monolithic libraries.

WEB TOOLS

DEFINITION:

- HTML.
- CSS.
- AngularJS.
- Bootstrap.
- PHP.
- NextJS.
- TypeScript.
- ExpressJS.

HTML:

It can be a stand-alone software dedicated to code writing and editing or a part of an IDE (Integrated Development Environment). An HTML editor provides more advanced features and is specifically designed for developers to create web pages more efficiently. It ensures every string of code is clean and works properly.

CSS:

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.

ANGULARJS:

AngularJS starts automatically when the web page has loaded.The **ng-app** directive tells AngularJS that the element is the "owner" of an AngularJS **application**.The **ng-model** directive binds the value of the input

field to the application variable **name**.The **ng-bind** directive binds the content of the <p> element to the application variable **name**

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js
"></script>
<body>
<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>
</body>
</html>
```


BOOTSTRAP:

Bootstrap is a free, open source front-end development framework for the creation of websites and web apps. Designed to enable responsive development of mobile-first websites, Bootstrap provides a collection of syntax for template designs.

PHP:

PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages. PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

```
<!DOCTYPE html>
<html>
<body>
<?php
echo "My first PHP script!";
```

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

NEXTJS:

Next.js enables you to create full-stack Web applications by extending the latest React features, and integrating powerful Rust-based JavaScript tooling for the fastest builds.

Next.js is an open-source web development React-based framework created by Vercel, which is famous for its unique features such as Server-side rendering and enhanced SEO. It has some additional features such as data fetching utilities, dynamic API routes, optimized builds, etc. It is a framework built upon React, Webpack, and Babel.

TYPESCRIPT:

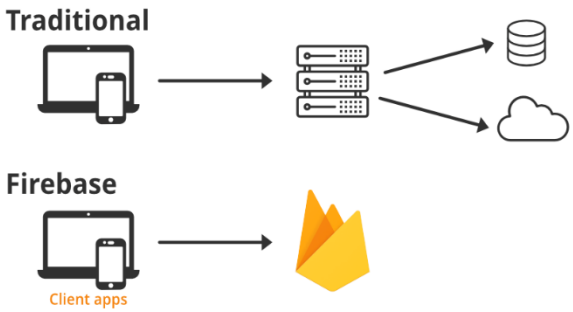
TypeScript allows specifying the types of data being passed around within the code, and has the ability to report errors when the types don't match. For example, TypeScript will report an error when passing a string into a function that expects a number. JavaScript will not.

EXPRESSJS:

Express. js is the most popular backend framework for Node. js, and it is an extensive part of the JavaScript ecosystem. It is designed to build single-page, multi-page, and hybrid web applications, it has also become the standard for developing backend applications with Node.

FIREBASE

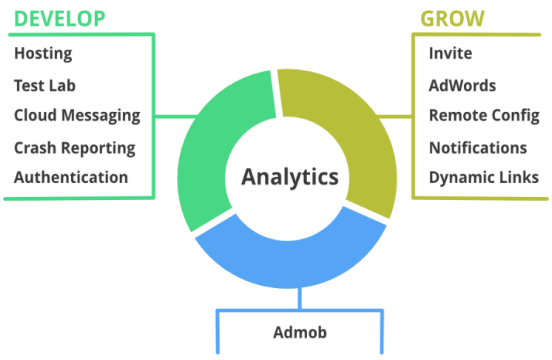
Firebase is a product of Google which helps developers to build, manage, and grow their apps easily. It helps developers to build their apps faster and in a more secure way. No programming is required on the firebase side which makes



it easy to use its features more efficiently. It provides services to android, iOS, web, and unity. It provides cloud storage. It uses NoSQL for the database for the storage of data.

Firebase initially was an online chat service provider to various websites through API and ran with the name **Envolve**. It got popular as developers used it to exchange application data like a game state in real time across their users more than the chats. This resulted in the separation of the Envolve architecture and its chat system.

Features of Firebase:

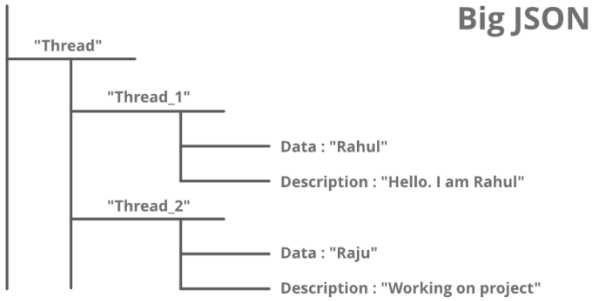


Mainly there are 3 categories in which firebase provides its services.

Build better applications

This feature mainly includes backend services that help developers to build and manage their applications in a better way. Services included under this feature are:

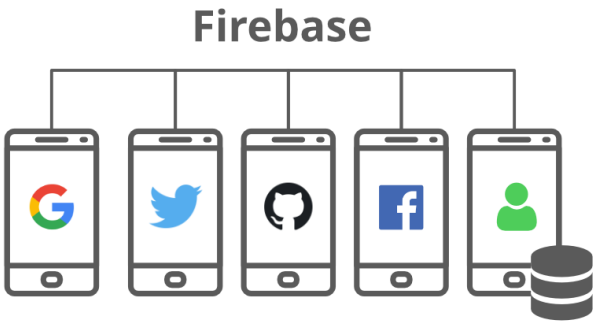
- **Realtime Database:** The Firebase Realtime Database is a cloud-based NoSQL database that manages your data at the blazing speed of milliseconds. In simplest term, it can be considered as a big JSON file.



- **Cloud Firestone:** The cloud Firestone is a NoSQL document database that provides services like store, sync, and query through the application on a global

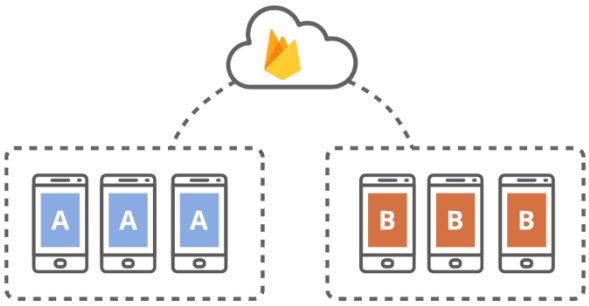
scale. It stores data in the form of objects also known as Documents. It has a key-value pair and can store all kinds of data like, strings, binary data, and even JSON trees.

- **Authentication:** Firebase Authentication service provides easy to use UI libraries and SDKs to authenticate users to your app. It reduces the manpower and effort required to develop and maintain the user authentication service. It even handles tasks like merging accounts, which if done manually can be hectic.



- **Remote Config:** The remote configuration service helps in publishing updates to the user immediately. The changes can range from changing components of the UI to changing the behaviour of the applications. These are often used while publishing seasonal offers and contents to the application that has a limited life.

- **Hosting:** Firebase provides hosting of applications with speed and security. It can be used to host Stati or Dynamic websites and microservices. It has the capability of hosting an application with a single command.
- **Firebase Cloud Messaging (FCM):** The FCM service provides a connection between the server and the application end users, which can be used to receive and send messages and notifications. These connections are reliable and battery-efficient.



Improve app quality:

Here majorly all the application performance and testing features are provided. All the features required to check and manage before launching your application officially are provided in this section. Services included are:

- **Crashlytics:** It is used to get real-time crash reports. These reports can further be used to improve the quality of the application. The most interesting part of this service is that it gives a detailed description of the crash which is easier to analyse for the developers.
- **Performance monitoring:** This service gives an insight to the performance characteristics of the applications. The performance monitoring SDK can be used to receive performance data from the application, review them, and make changes to the application accordingly through the Firebase console.
- **Test lab:** This service helps to test your applications on real and virtual devices provided by Google which are hosted on the Google Datacentres. It is a cloud-based app-testing infrastructure which supports testing the application on a wide variety of devices and device configurations
- **App Distribution:** This service is used to pre-release applications that can be tested by trusted testers. It comes in handy as decreases the time required to receive feedback from the testers.

Grow your app:

This feature provides your application analytics and features that can help you to interact with your user and make predictions that help you to grow your app. Services provided are:

- **Google analytics:** It is a Free app measurement service provided by Google that provides insight on app usage and user engagement. It serves unlimited reporting for up to 500 distinct automatic or user-defined events using the Firebase SDK.
- **Predictions:** Firebase Predictions uses machine learning to the application's analytics data, further creating dynamic user segments that are based on your user's behaviour. These are automatically available to use for the application through Firebase Remote Config, the Notifications composer, Firebase In-App Messaging, and A/B Testing.
- **Dynamic Links:** Deeps Links are links that directly redirects user to specific content. Firebase provides a Dynamic linking service that converts deep links into dynamic links which can directly take the user to a specified content inside the application. Dynamic links are used for converting web users to Native app users. It also increases the conversion of user-to-user sharing. In addition, it can also be used to integrate social media networks, emails, and SMS to increase user engagement inside the application.
- **A/B Testing:** It is used to optimize the application's experience by making it run smoothly, scaling the product, and performing marketing experiments.

Pros and Cons of Using Firebase:

Below listed are the advantages and disadvantages of using a Firebase backend:

Pros:

- Free plans for beginners.
- Real-time database is available.
- Growing Community.

- Numerous services are available.

Cons:

- It uses NoSQL so, people migrating from SQL might feel difficulty.
- It is still growing so; it is not tested to an extent.

Companies using Firebase

Below are some reputable organizations that rely on a firebase backend for its functioning:

- The New York Times
- Alibaba.com
- Gameloft
- Duolingo
- Trivago
- Venmo
- Lyft

DOCKER

Docker is a set of platforms as a service (PaaS) product that use Operating system-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than a virtual machine.

What is Docker?

Docker is an open-source containerization platform by which you can pack your application and all its dependencies into a standardized unit called a container. Containers are light in weight which makes them portable and they are isolated from the underlying infrastructure and from each other container. You can run the docker image as a docker container in any machine where docker is installed without depending on the operating system.

Docker is popular because of the following:

1. Portability.
2. Reproducibility.
3. Efficiency.
4. Scalability.

What is Docker file?

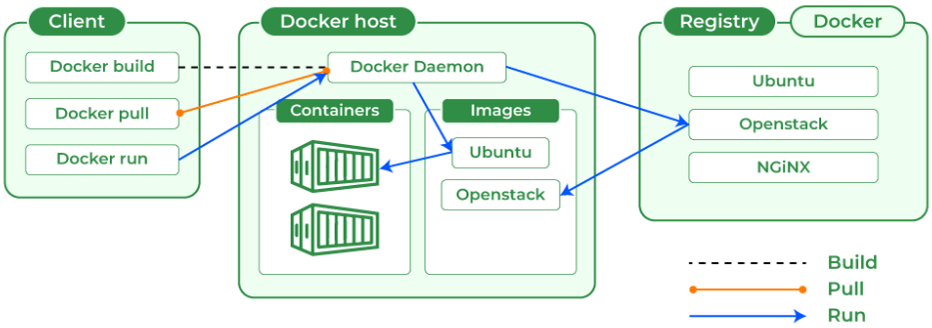
The Docker file uses DSL (Domain Specific Language) and contains instructions for generating a Docker image. Docker file will define the processes to quickly produce an image. While creating your application, you should create a Docker file in order since the Docker daemon runs all of the instructions from top to bottom.

- It is a text document that contains necessary commands which on execution help assemble a Docker Image.
- Docker image is created using a Docker file.

How Docker Works

Docker makes use of a client-server architecture. The Docker client talks with the docker daemon which helps in building, running, and distributing the docker containers. The Docker client runs with the daemon on the same system or we can connect the Docker client with the Docker daemon remotely. With the help of REST API over a UNIX socket or a network, the docker client and daemon interact with each other. To know more about working of docker refer to the

Architecture of Docker.



What is Docker Image?

It is a file, comprised of multiple layers, used to execute code in a Docker container. They are a set of instructions used to create docker containers. Docker Image is an executable package of software that includes everything needed to run an application. This image informs how a container should instantiate, determining which software components will run and how. Docker Container is a virtual environment that bundles application code with all the dependencies required to run the application. The application runs quickly and reliably from one computing environment to another.

What is Docker Container?

Docker container is a runtime instance of an image. Allows developers to package applications with all parts needed such as libraries and other dependencies. Docker Containers are runtime instances of Docker images. Containers contain the whole kit required for an application, so the application can be run in an isolated way. For e.g.- Suppose there is an image of Ubuntu OS with NGINX SERVER when this image is run with the docker run command, then a container will be created and NGINX SERVER will be running on Ubuntu OS.

What is Docker Hub?

Docker Hub is a repository service and it is a cloud-based service where people push their Docker Container Images and also pull the Docker Container

Images from the Docker Hub anytime or anywhere via the internet. It provides features such as you can push your images as private or public. Mainly DevOps team uses the Docker Hub. It is an open-source tool and freely available for all operating systems. It is like storage where we store the images and pull the images when it is required. When a person wants to push/pull images from the Docker Hub they must have a basic knowledge of Docker. Let us discuss the requirements of the Docker tool.

What is Docker Compose?

Docker Compose will execute a YAML-based multi-container application. The YAML file consists of all configurations needed to deploy containers Docker Compose, which is integrated with Docker Swarm, and provides directions for building and deploying containers. With Docker Compose, each container is constructed to run on a single host.

Docker Commands

There is “n” no. of commands in docker following are some of the commands mostly used.

1. Docker Run
2. Docker Pull
3. Docker PS
4. Docker Stop
5. Docker Start
6. Docker rm
7. Docker RMI
8. Docker Images
9. Docker exec
10. Docker Login

Docker Engine

The software that hosts the containers is named Docker Engine. Docker Engine is a client-server-based application. The docker engine has 3 main components:

1. **Server:** It is responsible for creating and managing Docker images, containers, networks, and volumes on the Docker. It is referred to as a daemon process.
2. **REST API:** It specifies how the applications can interact with the Server and instructs it what to do.
3. **Client:** The Client is a docker command-line interface (CLI), that allows us to interact with Docker using the docker commands.

Why use Docker?

Docker can be used to pack the application and its dependencies which makes it lightweight and easy to ship the code faster with more reliability. Docker make every simple to run the application in the production environment docker container can be platform independent if the docker engine is installed in the machine.

What is Docker for AWS?

Docker is the most powerful tool to run the application in the form of containers. Docker container is light in weight and can be run on any operating system's provides the Amazon Elastic Container Service (Amazon ECS) it is a fully managed container service by which you can deploy, scale and manage the docker containers. Amazon ECS is the most reliable platform according to the performance and also it can be integrated with the other AWS Service like load balancing, service discovery, and container health monitoring. To know more about Amazon Elastic Container Service (Amazon ECS).

Difference Between Docker Containers and Virtual Machines

Docker Containers	Virtual Machines
Docker Containers contain binaries, libraries, and configuration files along with the application itself.	Virtual Machines (VMs) run on Hypervisors, which allow multiple Virtual Machines to run on a single machine along with its own operating system.
They don't contain a guest OS for each container and rely on the underlying OS kernel, which makes the containers lightweight.	Each VM has its own copy of an operating system along with the application and necessary binaries, which makes it significantly larger and it requires more resources.
Containers share resources with other containers in the same host OS and provide OS-level process isolation.	They provide Hardware-level process isolation and are slow to boot.

NODE JS

- Node.js is an open-source server-side runtime environment built on Chrome's V8 JavaScript engine.
- It provides an event driven ,non-blocking (synchronous)I/O and cross-platform run time environment for building highly scalable server-side application using JavaScript.

- Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

Main feature of Node.js is:

- Event driven Architecture
- Non Blocking I/O model
- Asynchronous by default

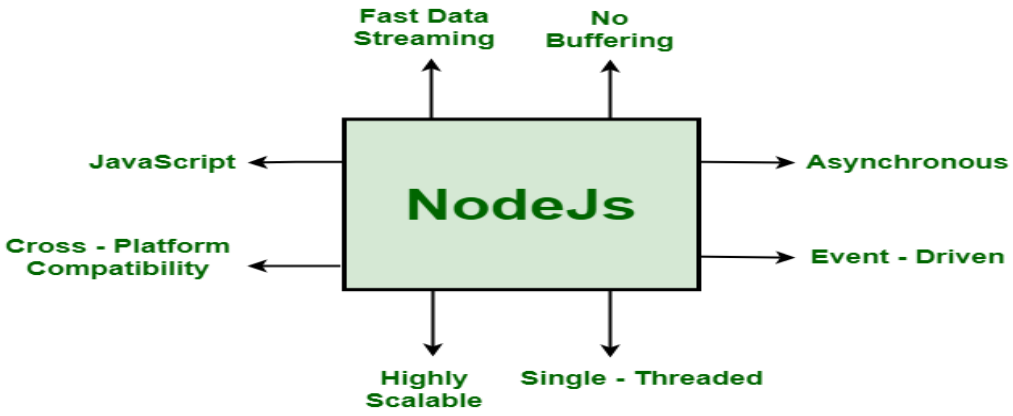
What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

What is a Node.js File?

- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

Key Features of NodeJs



Asynchronous and Event-Driven: The Node.js library's APIs are all asynchronous (non-blocking) in nature. A server built with Node.JS never waits for data from an API. from an API. After accessing an API, the server moves on to the next one. In order to receive and track responses of previous API requests, it uses a notification mechanism called Events.

Single-Threaded: Node.js employs a single-threaded architecture with event looping, making it very scalable. In contrast to typical servers, which create limited threads to process requests, the event mechanism allows the node.js server to reply in a non-blocking manner and makes it more scalable. When compared to traditional servers like Apache HTTP Server, Node.js uses a single-threaded program that can handle a considerably larger number of requests.

Scalable: NodeJs addresses one of the most pressing concerns in software development: scalability. NodeJs can also handle concurrent requests efficiently. It has a cluster module that manages load balancing for all CPU cores that are active. The capability of NodeJs to partition applications horizontally is its most appealing feature. It achieves this through the use of child processes. This allows the organizations to provide distinct app versions to different target audiences, allowing them to cater to client preferences for customization.

Quick execution of code: Node.js makes use of the V8 JavaScript Runtime motor, which is also used by Google Chrome. Hub provides a wrapper for the JavaScript motor, which makes the runtime motor faster. As a result, the preparation of requests inside Node.js becomes faster as well.

Cross-platform compatibility: NodeJS may be used on a variety of systems, including Windows, Unix, Linux, Mac OS X, and mobile devices. It can be paired with the appropriate package to generate a self-sufficient executable.

Uses JavaScript: JavaScript is used by the Node.js library, which is another important aspect of Node.js from the engineer's perspective. Most of the engineers are already familiar with JavaScript. As a result, a designer who is familiar with JavaScript will find that working with Node.js is much easier.

Fast data streaming: When data is transmitted in multiple streams, processing them takes a long time. Node.js processes data at a very fast rate. It processes and uploads a file simultaneously, thereby saving a lot of time. As a result, NodeJs improves the overall speed of data and video streaming.

No Buffering: In a Node.js application, data is never buffered.

File Handling in Nodejs:

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

REACT

- React is a free and open-source front-end JavaScript library for building user interfaces based on components.
- It is maintained by Meta and a community of individual developers and companies.
- React can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js.
- React is a JavaScript-based UI development library.
- Although React is a library rather than a language, it is widely used in web development

Why React?

React's popularity today has eclipsed that of all other front-end development frameworks. Here is why:

- Easy creation of dynamic applications
- Improved performance
- Reusable componenents
- Unidirectional dataflow
- Small learning curve
- It can be used for the development of both web and mobile apps
- Dedicated tools for easy debugging

ReactJS Advantages

1. React.js builds a customized virtual DOM. Because the JavaScript virtual DOM is quicker than the conventional DOM, this will enhance the performance of apps.
2. ReactJS makes an amazing UI possible.
3. Search - engine friendly ReactJS.
4. Modules and valid data make larger apps easier to manage by increasing readability.
5. React integrates various architectures.
6. React makes the entire scripting environment process simpler.
7. It makes advanced maintenance easier and boosts output.
8. Guarantees quicker rendering
9. The availability of a script for developing mobile apps is the best feature of React.
10. ReactJS is supported by a large community.

Limitations

1. Only addresses the app's angle and distance; as a result, additional techniques must be selected if you want a full collection of development tools.
2. Employs inline scripting and JSX, which some programmers might find uncomfortable.

Features of React

- Re-usability
- Nested components
- Render method
- Passing properties

React offers some outstanding features that make it the most widely

adopted library for frontend app development.

JSX is a JavaScript syntactic extension. It's a term used in React to describe how the user interface should seem. You can write HTML structures in the same file as JavaScript code by utilizing JSX.

```
const name = 'Simplilearn';
const greet = <h1>Hello, {name}</h1>;
```

The above code shows how JSX is implemented in React. It is neither a string nor HTML. Instead, it embeds HTML into JavaScript code.

Architecture

In a Model View Controller(MVC) architecture, React is the 'View' responsible for how the app looks and feels.

Here are some of the most popular React frameworks:

- *React Router:*

  React Router is a library that helps you manage the navigation in your React applications. It provides a variety of components that make it easy to create complex routing patterns.

- *Redux:*

  Redux is a state management library that helps you manage the state of your React applications. It provides a simple, predictable way to store and update data in your applications.

- *React Native:*

  React Native is a framework that lets you build native mobile apps with React.js. It is essentially a cross-platform framework that allows you to use the same code to build apps for both iOS and Android.

DJANGO

Django is a free and open-source Python web framework that encourages rapid development and clean, pragmatic design. It takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel.

Django and AngularJS in relation to each other:

Front-end vs. back-end: Django is a back-end framework, while AngularJS is a front-end framework. This means that Django is used to build the server-side logic of a web application, while AngularJS is used to build the user interface.

Complementary technologies: Django and AngularJS can be used together to build powerful web applications. For example, Django can be used to handle routing, database management, and authentication, while AngularJS can be used to create interactive and dynamic user interfaces.

Difference between Django and Angular JS:

Django

- Purpose: Full-stack Python web framework
- Language: Python
- Strengths: Rapid development, security, scalability, large community
- Weaknesses: Can be complex for beginners, not as flexible as some other frameworks

AngularJS

- Purpose: Front-end JavaScript framework for building single-page applications (SPAs)
- Language: JavaScript
- Strengths: Fast performance, data binding, modularity, large community
- Weaknesses: Can be complex for beginners, not as well-suited for backend development

Django's key features include:

- Object-relational mapper (ORM): Django provides a powerful ORM that makes it easy to interact with your database.
- Admin interface: Django includes a built-in admin interface that makes it easy to manage your app's data.
- Authentication and authorization: Django provides built-in support for user authentication and authorization.
- Template system: Django's template system makes it easy to create dynamic HTML pages.
- Caching: Django has a built-in caching system that can improve the performance of your app.
- Internationalization and localization: Django supports internationalization and localization, so you can easily build multilingual websites and apps.

Django is a popular choice for building a wide range of web applications, including:

- Content management systems (CMSs)
- E-commerce websites
- Social networking sites
- News websites

- Business websites
- Educational websites

Benefits of using Django:

- Rapid development: Django is designed for rapid development. It provides a number of features that make it easy to get started quickly and build complex applications quickly.
- Security: Django is designed to be secure. It includes a number of security features out of the box, such as cross-site scripting (XSS) protection and SQL injection protection.
- Scalability: Django is scalable. It can be used to build small websites as well as large, enterprise-grade applications.
- Large community: Django has a large and active community. This means that there are a lot of resources available to help you learn Django and get help if you need it.

UI/UX DESIGN

UI stands for user interface. It refers to the visual elements of a website or application, such as the buttons, menus, and forms. The UI is what users interact with, so it is important that it is easy to use and understand.

UX stands for user experience. It refers to the overall experience that a user has when using a website or application. This includes the UI, but it also includes factors such as usability, performance, and customer support. A good UX makes it easy for users to find what they are looking for and complete their tasks.

UI/UX and AngularJS

UI/UX and AngularJS are both important aspects of building web applications. UI/UX is about making sure that the application is easy to use and understand,

while AngularJS is a framework that can be used to build SPAs that provide a fluid and responsive user experience.

Relationship between UI/UX and AngularJS:

UI/UX and AngularJS are closely intertwined in the development of web applications. UI/UX principles guide the design of the application's interface, ensuring that it is intuitive, user-friendly, and aligns with the overall user experience goals. AngularJS, on the other hand, provides the tools and frameworks to implement these UI/UX design decisions, enabling developers to create responsive, interactive, and visually appealing user interfaces.

Here are some of the ways that UI/UX and AngularJS can be used together to build better web applications:

- Use AngularJS to create dynamic and responsive UIs: AngularJS makes it easy to create UIs that are responsive to user input. This can create a more fluid and enjoyable user experience.
- Use AngularJS to improve performance: AngularJS can help to improve the performance of web applications by caching data and reducing the number of server requests. This can make the application feel more responsive to users.
- Use AngularJS to make it easier to maintain code: AngularJS's data binding and directive features can make it easier to maintain complex UIs. This can save time and money in the long run.

How AngularJS contributes to UI/UX:

- Data binding: AngularJS's two-way data binding mechanism simplifies data synchronization between the user interface and the underlying application data, ensuring that the interface reflects the current state of the data and vice versa.

- Directives: AngularJS provides a collection of directives, which are HTML attributes that extend the functionality of HTML elements. These directives enable developers to create custom components and behaviors, enhancing the flexibility and richness of the user interface.
- Routing: AngularJS's routing system facilitates seamless navigation between different pages or sections of the application, ensuring a smooth and consistent user experience.
- Templating: AngularJS's templating engine allows developers to separate the presentation layer (HTML) from the application logic, making it easier to maintain and extend the user interface.
- Testing: AngularJS's robust testing framework enables developers to write unit tests and end-to-end tests, ensuring that the user interface functions as intended and provides a consistent and bug-free experience.

In summary, UI/UX and AngularJS work in tandem to create web applications that are not only visually appealing and easy to use but also meet the specific needs and expectations of their users. UI/UX principles guide the design of the user interface, while AngularJS provides the tools and frameworks to implement these design decisions effectively, resulting in a positive and engaging user experience.

****