

UNIT-I

Introduction to Web Technology

Web Page

- A web page is a document available on World Wide Web.
- Web Pages are stored on **web server** and can be viewed using a **web browser**.
- A web page can contain huge information including text, graphics, audio, video and hyper links. These hyper links are the link to other web pages.
- Each individual web page, image, and video is identified by a distinct **Uniform Resource Locator** (URL), enabling browsers to retrieve these resources from a **web server** and display them on the **user's** device.
- **URL** is an acronym for Uniform Resource Locator and is a reference (an address) to a resource on the Internet.
 - A URL has two main components

1. Protocol identifier: <http://gmail.com>

The protocol identifier is **http**

2. Resource name: <http://gmail.com>

The resource name is **gmail.com**

Eg: <http://www.amazon.in/mobiles.html>

Web Browser

A **web browser** (commonly referred to as a **browser**) is a **software application** for accessing information on the **World Wide Web**.

The most popular browsers are Chrome, Firefox, Safari, Internet Explorer, Microsoft Edge, opera and UC Browser.

Collection of linked web pages on a web server is known as **website**.

Web Server

A web server is a system that delivers content or services to end users over the internet. A web server consists of a physical server, server operating system and software used to facilitate HTTP communication.

Web server is a program that uses HTTP to serve files that create web pages to users in response to their requests, which is sent by their computers HTTP connection.

- A web server is also known as an internet server.
- Web servers are core for any web hosting.
- Most of the web hosting companies select web servers based on clients requirement, the number of clients on a single server, the applications/software clients use and the amount of traffic they generate that could handle by a web server. So, choose the web server which meets the requirements.



Any server that delivers an XML document to another device can be a web server. A better definition might be that a web server is an Internet server that responds to HTTP requests to deliver content and services.

Always a web server is connected to the internet. Every web server that connects to the Internet will be having a unique address.

Hosting companies use different web servers considering the requirements of their clients.

1. Apache web server

One of the most popular web servers in the world developed by the **Apache Software Foundation**.



- Apache is open source software which supports almost all operating systems including Linux, UNIX, Windows, FreeBSD, Mac OS X and more.
- Customization of apache web server is easy as it contains a modular structure. It is also an open source which means that you can add your own modules to the server when to require and make modifications that suit your requirements.
- It is more stable than any other web servers and is easier to solve administrative issues. It can be installed on multiple platforms successfully.
- Handles multiple requests simultaneously using multithreading.

2. IIS web server

IIS is a Microsoft product (Internet Information Services).

- Microsoft developed this product and they maintains, thus it works with all the windows operating system platforms.
- But it is not an open source and more over adding personal modules is not easy and modification becomes a little difficult job.



3. Nginx web server

Another free open source web server is Nginx. It is known for its high performance, stability, simple configuration and low resource usage.

- This web server doesn't use threads to handle requests rather a much more scalable event-driven architecture which uses small and predictable amounts of memory under load.

4. LightSpeed web server

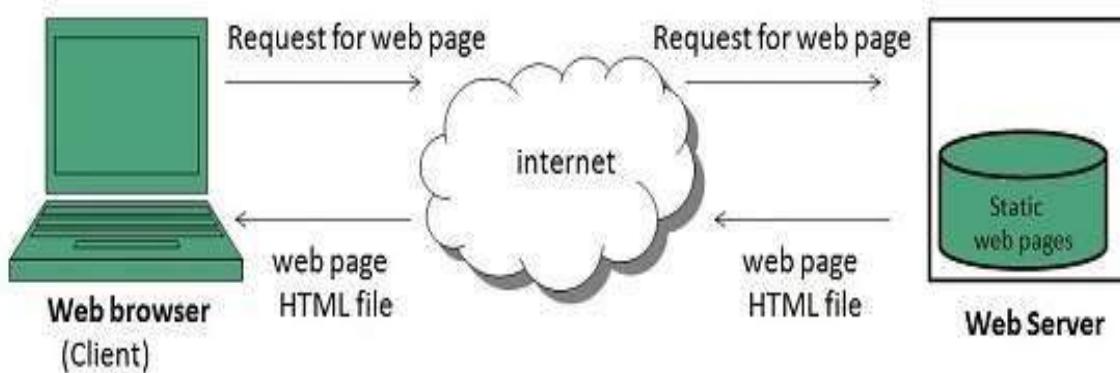
LiteSpeed (LSWS) is a high-performance Apache drop-in replacement. LSWS is the most popular web server on the internet and it is a commercial web server.

- Upgrading your web server to LiteSpeed will improve performance and lower operating costs.

Types of Web Pages

1. Static web pages

- **Static web pages** are also known as flat or stationary web page.
- They are loaded on the client's browser as exactly they are stored on the web server. Such web pages contain only static information.
- User can only read the information but can't do any modification or interact with the information.
- Static web pages are created using only **HTML and CSS**.
- Static web pages are only used when the information is no more required to be modified.



2. Dynamic Web pages

- **Dynamic web page** shows different information at different point of time.
- It is possible to change a portion of a web page without loading the entire web page.
- Allows to create interactive web pages.

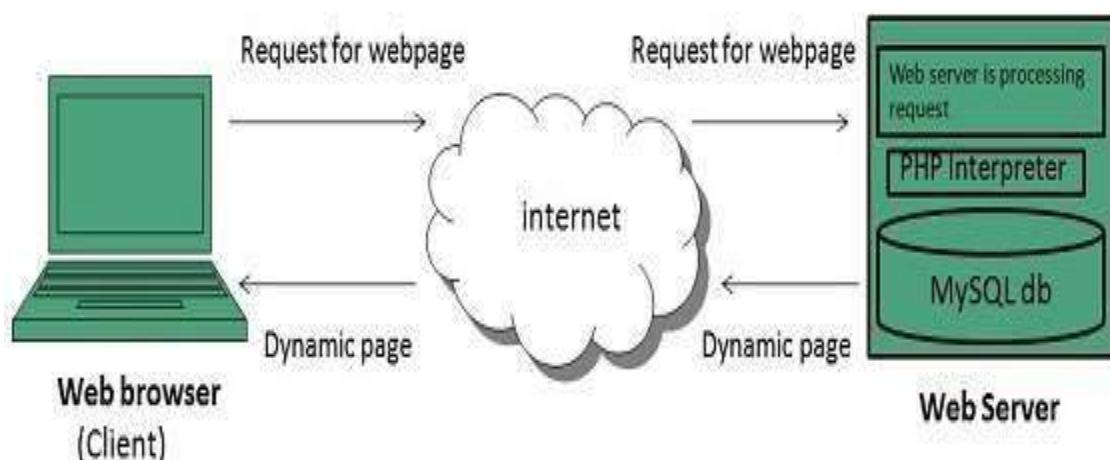
I) SERVER-SIDE DYNAMIC WEB PAGE

It is created by using server-side scripting.

There are server-side scripting parameters that determine how to assemble a new web page which also includes setting up of more client-side processing.

II) CLIENT-SIDE DYNAMIC WEB PAGE

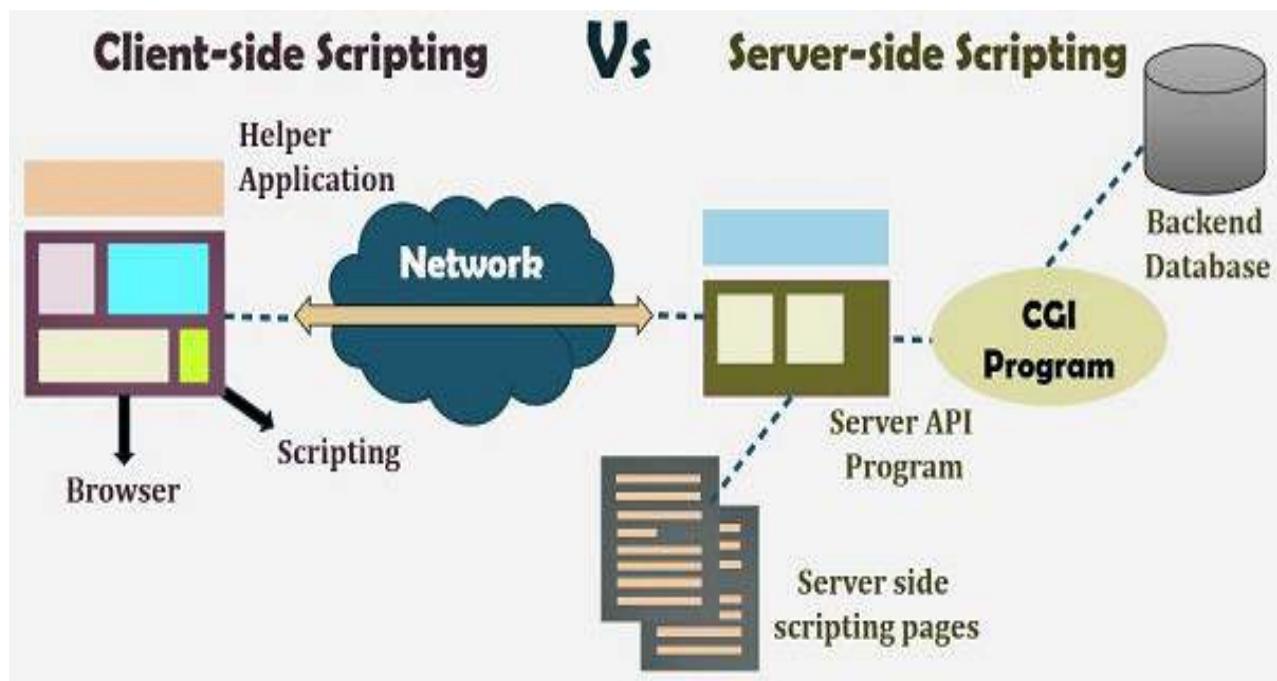
It is processed using client side scripting such as JavaScript. And then passed in to **Document Object Model (DOM)**.



Scripting Languages

Server-side Scripting and Client-side Scripting

A script is generally a series instruction, which has to be executed on other program or application.



The scripts can be written in two forms

1. Server Side Scripting

- It is a technique of programming for producing the code which can run software on the server side.
- Any scripting or programming that can run on the web server is known as server-side scripting.
- Script Code Written on Server machine
- The server side scripting involves server for its processing
- A server-side script is executed in the server end which users cannot see.

- The operations like customization of website, dynamic change in website content, response generation to the user queries, accessing the database, making simple calculations etc.
- The server side scripting constructs a communication link between a server and client.
- The web server abstracts the scripts from the end user, which makes the data and source code more secure.
- PHP, Python, Ruby, JSP, ASP, ASP.Net, ColdFusion etc are Server side scripting languages.

2. Client side Scripting

- Script code written on client. (Browser)
- It is performed to generate a code that can run on the client end (Browser) without needing the server side processing.
- Script is placed inside the HTML document.
- Used to examine the user's form for the errors before submitting it.
- The effective client-side scripting can significantly reduce the server load.
- Client-side scripting requires browsers to run the scripts on the client machine but does not interact with the server while processing the client-side scripts.
- HTML, CSS, JavaScript, Jscript, VB Script etc are client – side scripting languages.
- The client-side script executes the code to the client side which is visible to the users.

COMPARISON	SERVER-SIDE SCRIPTING	CLIENT-SIDE SCRIPTING
Basic	Works in the back end which could not be visible at the client end.	Works at the front end and script are visible among the users.
Processing	Requires server interaction.	Does not need interaction with the server.
Languages involved	PHP, Java, JSP, ASP.net, ColdFusion, Python, etc.	HTML, CSS, JavaScript, VB Script, etc.
Affect	Could effectively customize the web pages and provide dynamic websites.	Can reduce the load to the server.
Security	Relatively secure.	Insecure
Scalability	More Flexible	Less Flexible

Plug-ins

A plug-in (or plugin, add-in, addin, add-on)

Is a software component that adds a specific feature to an existing computer program. When a program supports plug-ins, it enables customization.

Web browsers have historically allowed executables as plug-ins, though they are now mostly deprecated. (These are a different type of software module than browser extensions.) . Two plug-in examples are the Adobe Flash Player for playing videos and a Java virtual machine for running applets.

Applications support plug-ins for many reasons. Some of the main reasons include:

- to enable third-party developers to create abilities which extend an application
- to support easily adding new features
- to reduce the size of an application
- To separate **source code** from an application because of incompatible **software licenses**.

Types of applications and why they use plug-ins:

- **Audio editors** use plug-ins to generate process or analyze sound. **Ardour** and **Audacity** are examples of such editors.
- **Email clients** use plug-ins to decrypt and encrypt email. **Pretty Good Privacy** is an example of such plug-ins.
- **Graphics software** use plug-ins to support file formats and process images. (*c.f.* **Photoshop** plug-in)
- **Media players** use plug-ins to support file formats and apply filters. **foobar2000**, **GStreamer**, **Quintessential**, **VST**, **Winamp**, **XMMS** are examples of such media players.
- **Text editors** and **Integrated development environments** use plug-ins to support **programming languages** or enhance development process *e.g.*, **VisualStudio**, **RADStudio**, **Eclipse**, Visual Studio itself can be plugged into other applications via **Visual Studio Tools for Office** and **Visual Studio Tools for Applications**.
- **Web browsers** have historically used **executables** as plug-ins, though they are now mostly **deprecated**. Examples include **Adobe Flash Player**, **Java SE**

Tiers

A “tier” is referred to as a “layer”. The three Layers present in n-tier architectures are

A *Presentation Layer* that sends content to browsers in the form of HTML/CSS. Static or dynamically generated content rendered by the browser (**front-end**). It is also called client layer.

An *Application Layer* that uses an application server and processes the business logic for the application. A dynamic content processing and generation level Application server. This might be written in C#, Java, C++, Python, Ruby, etc. It is also called Logical Layer. This layer acts as a mediator between the Presentation and the Database layer. Complete business logic will be written in This layer. It is also called as Middleware.

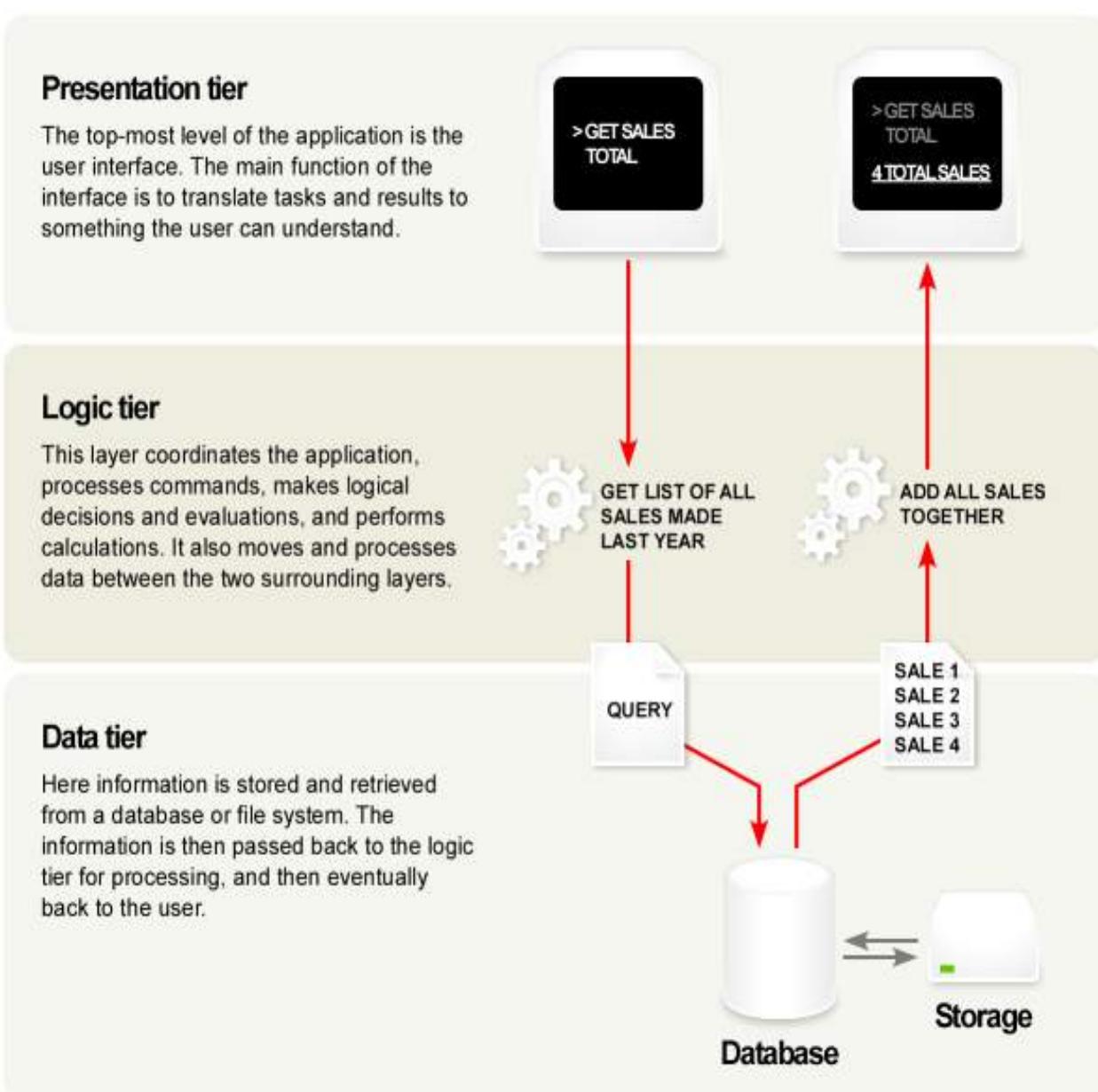
A *Data Layer* which is a database management system that provides access to application data. A database comprising both data sets and RDBMS software that manages and provides access to the data (back-end). This could be MSSQL, MySQL, Oracle etc. It contains methods that connects the database and performs required action. e.g.: insert, update, delete etc.

Three-tier architecture provides numerous benefits in web application development. It allows a developer the opportunity to extend, modularize, and be able to configure their application easily.

Example: As a simple example, to book a movie ticket using web application

1. The **presentation layer** displays a web page with some fields for you to enter, like the date to view the movie and your zip code.

2. This information is then passed to the **application layer**, which formats a query and passes it to the **database layer**.
3. The database system runs the query and returns the results (**a list of movies available within your geographic area**) to the application layer, which formats it into a web page. The page is then sent back to the browser, where the presentation layer displays it on a laptop or other device.

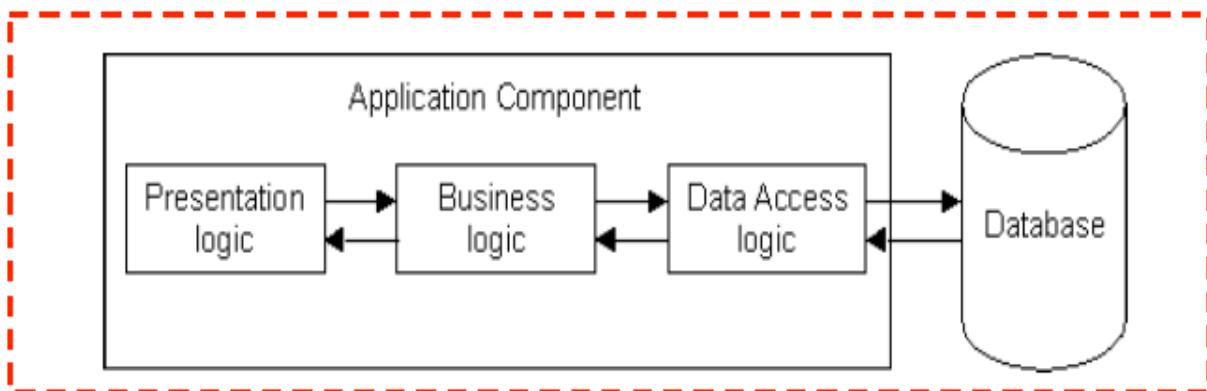


The benefits of separating an application into tiers:

- It gives the ability to **update the technology** stack of one tier, without impacting other areas of the application.
- It allows for different development teams to each work on their **own areas of expertise**. Today's developers are more likely to have deep competency in one area, like coding the front end of an application, instead of working on the full stack.
- You are able to **scale** the application up and out. A separate back-end tier, for example, allows you to deploy to a variety of databases instead of being locked into one particular technology. It also allows you to scale up by adding multiple web servers.
- It adds **reliability** and more **independence** of the underlying servers or services.
- It provides an **ease of maintenance** of the code base, managing presentation code and business logic separately, so that a change to business logic, for example, does not impact the presentation layer.

The various architectures are

1. 1- Tier Architecture

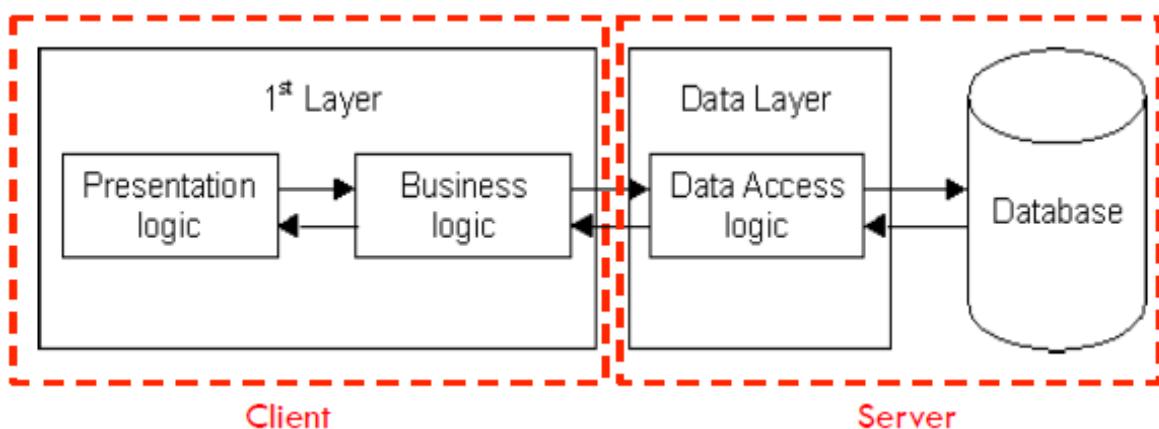


- All the three layers are on the same machine.
- Presentation, Business Logic and Data access logic are tightly connected
- Easy and quick to develop
- Useful for small offices

Limitations

- Difficult to upgrade
- Not scalable. i.e. Hard to increase volume of processing **(Scalability)**
- Moving to a new machine requires rewriting everything. **(Portability)**
- Did not protect valuable "Business Logic".
- Changing of one layer needs to change others. **(maintanance)**

2. 2-Tier Architecture

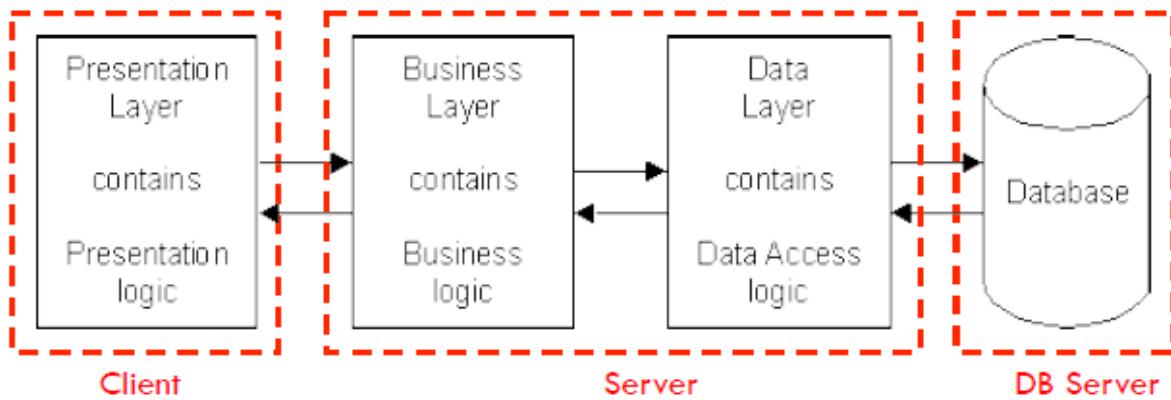


- Database runs on Server
- Presentation and logic layers still tightly connected.
- Protects business logic from UI

Limitations

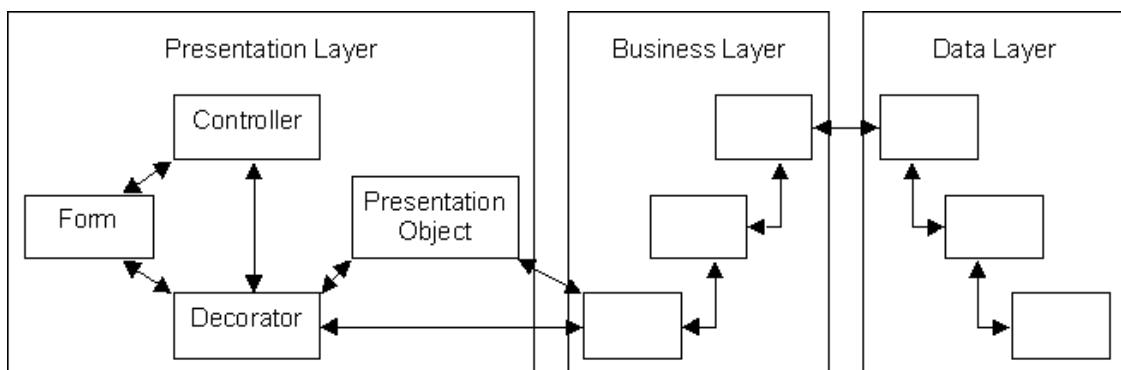
- Business-logic is implemented on the PC
- Increased network traffic
- Application logic can't be reused
- Must design/implement protocol for communication between client and server.

3. 3-Tier Architecture



- Each layer can potentially run on a different machine
- Clear separation of user-interface-control and data presentation from application-logic.
- Presentation, logic, data layers disconnected
- Change in business logic won't need change in other layers.
- Dynamic load balancing by use of multiple servers
- Easier to maintain
- Components are reusable
- Faster development. i.e.
 - Web designer does presentation
 - Software engineer does logic
 - DB admin does data model

4. N-Tier Architecture



- The tiers may contain one or more components of the application.
- The components in one tier can communicate ONLY with the components in the tiers above and below.
- The components implement major functionalities in the application

Advantages

- Easy to reuse
- Easy to develop
- Easy to change
- Easy to manage

Limitations

- In small applications, the benefits are usually not visible.
- Every data that goes in the system from the user to the database must pass through the components in the middle layers, and therefore the response time of the system will be slower.

Introduction To HTML

Date _____
Page _____

HTML - Hypertext Markup Language is most commonly used markup language to create web pages.

Hypertext - Refers to the way in which webpages are linked together. The link available on a webpage is called hypertext.
Navigation through the pages is not linear using hypertext.

"HTML allows to format, arrange and group text, display text as links, add images and multimedia to a webpage. It allows to create and work with style sheets, controls webpages, and embed scripting language code in a webpage."

HTML was created by Berners-Lee.

HTML Characteristics

1. World Wide Web consortium (W3C) is an organization defines standards and specifications for HTML and is responsible for updatations.
2. HTML is used to create static webpages.
3. HTML is platform independent, but browser dependant language.
4. HTML is a markup language, but not a programming language.
5. HTML has no data-types, memory and control statements.
6. HTML is written in the form of tags, which are a pair of angle brackets '< and >' and some text placed between these brackets.
7. HTML is not case sensitive (Case Insensitive) FF markup language.
8. A HTML document is created by using any text editor like notepad, notepad++, Editplus etc.
9. A HTML file is saved with file name extension with ".htm" or ".html".
10. The current Version of HTML is HTML 5.0 (2012)
11. HTML with CSS & Javascript used to create Dynamic webpages.
12. Most of the HTML elements can be nested.

13. A HTML file is executed with an enabled browser.
14. A browser carry "HTML Interpreter" to process HTML Tags / elements.
15. HTML elements are building blocks of HTML pages.
16. Browsers do not display HTML tags, ~~but~~ but use them to render the content of the page.

Basic Structure of HTML

The Basic HTML document structure is

```
<!DOCTYPE HTML "-//W3C//DTD HTML 5.0//EN">
<html>
  <head>
    <title> Title of the webpage </title>
  </head>
  <body>
    Contents to present on webpage
  </body>
</html>
```

- HTML is written in the form of tags. (`head`, `title`)
- Tags are pairs of angle brackets '`<`' and '`>`' and some text is placed between these brackets.
- The text present in pair of angle brackets is defines "HTML element".
- A HTML element can be
 - ① Container tag — The element contains some text (or) other HTML elements.

e.g.: `<title> html page </title>`

- uses both opening and closing tag.

opening tag - `<Element-name>`

closing tag - `</Element-name>`

2. Non-Container Tag - The element has no content.

Eg: Horizontal Row (<HR>)

Break (
) etc.

- Do not need a closing tag.

Every HTML element has two properties.

① Attributes

② Contents

<Element-name attribute-name = "attribute-value" >

Content

</Element-name>

Eg:

HTML document

Attributes → These are name-value pairs

→ Attribute values are specified in single and double quotes.

→ placed within open tag

Contents - plain text or other HTML elements.

* Most of the HTML elements are nested

* Attributes of an HTML element are optional.

<!DOCTYPE > → Specifies the Document type and HTML version.

Also defines "Document Type Definition" (Grammar) used by the document.

The Basic HTML elements are

1. `<html>` element (`<html>`)

- * `<html>` element starts an HTML document.
- * `<HTML>` element encloses the complete HTML document.
It contains / comprises of two elements.

- ① `<head>` - document header
- ② `<body>` - document body

The attributes of `<html>` element are

`class` - Represents the class of element
and is used to render the content.

`dir` - Direction of text.

i.e. `ltr` - left to right text direction

`rtl` - right to left text direction

Eg: `<html dir = "rtl" >`

`id` - unique alphanumeric identifier

`lang` - base language used for element

`xmlns` - Declares a name space for the
custom tags used in HTML document.

Eg: `<html id = "20" dir = "rtl" >`

≡

`</html >`

2. `<head>` element

Represents document header. It contains the general information about the HTML document.

The information includes

- title of webpage
- keywords for search engine
- Base address for URLs

The attributes of `<head>` element are

`class` - Represents the class of Element, used to

`dir` - Direction of text Render content

`id` - Unique alphanumeric identifier

`lang` - Base language used for element

`title` - Holds additional information for element

`style` - Represents inline styles indicating how to render the element.

The elements to be added to `<head>` element

`<base>` - Base URL for the document

`<basefont>` - Base font for the document

`<bgsound>` - Background sound

`<link>` - A Relation between document & another object.

`<meta>` - Represents header information

`<title>` - Title of webpage that appears in the web browser.

- <script> — Holds programming script statements in VBScript, Javascript etc.
- <style> — To Represent Embedded style for rendering content.

3. <title> element

- It contains title of HTML document,
- Appears on the title bar of the web browser
- Used by search engines to refer the document.
- for simplicity upto 256 chars used in title.

The attributes of <TITLE> element

- class — class of element, used to render content
- id — unique alphanumeric identifier
- lang — base language used for element
- style — inline styles indicating how to render the content.

Eg:

```
<head id = 10 dir = "ltr">  
    <title> first html page </title>  
    <meta name = "Author" content = "Ramesh" >  
    <meta name = "Keywords" content = "html" >  
    <meta name = "Description" content = "Hello" >  
    <basefont color = "green" size = "8" >  
    <bgsound src = "Pathof-Audio file" >  
</head >
```

4. <body> element

- Represents the document body of HTML document
- It includes entire content that appear on the Web browser.
- It also includes text, images and multimedia elements.
- All other HTML elements present in <body>.

The attributes of <body> element are

- alink** — color of the hyperlinks when they are clicked
- vlink** — color of the hyperlinks that have been visited.
- background** — URL of the graphic file used as the background of the browser.
- bgcolor** — color of the background of the browser.
- bottommargin** — empty space at the bottom of the document in pixels.
- bgproperties** — background will not scroll if the text scrolls, when the value is "fixed".
- class** — class of the element & used to render content
- dir** — direction of text
- id** — unique alphanumeric identifier
- lang** — base language used for element
- language** — Represents scripting language used for element
- link** — color of hyperlinks that have not yet been visited.
- leftmargin** — empty space to the left of the document in pixels

rightmargin — empty space to the right of document
in pixels.

marginheight — height of top and bottom margins
(in pixels)

marginwidth — width of left & right margin.

topmargin — Empty space at the
top of document in pixels
(in pixels).

text — color of document text in
the document.

title — holds additional information for
the element, such as tooltips.

Eg:

```
<body text = "yellow" title = "welcome"  
leftmargin = 100 bgcolor = "green"  
vlink = "#eefff2" >
```

COMMON TAGS

Date _____
Page _____

1 Creating Headings on a Webpage

- HTML defines special, pre-formatted tags to create headings on a webpage.
- Headings appears by default bold and their size depends on their level.
- Headings get their own line on the webpage by starting in a new line.
- Headings define format and structure of a document.
- HTML defines 6 heading tags (Container tags)
 $\langle H_1 \rangle, \langle H_2 \rangle, \langle H_3 \rangle, \langle H_4 \rangle, \langle H_5 \rangle, \langle H_6 \rangle$
Large to small size. →

The attributes of heading tags are

align - To change the alignment of text. The value can be

center - Aligns the whole text to center

left - Aligns the whole text to left

right - Aligns the whole text to right

justify - Justifies whole text & indents the first line.

Eg:

$\langle h_1 \rangle$ Welcome to HTML $\langle /h_1 \rangle$

$\langle h_2$ align = "center" \rangle Welcome to HTML $\langle /h_2 \rangle$

$\langle h_3$ align = "justify" \rangle Welcome to HTML $\langle /h_3 \rangle$

$\langle h_3$ align = "justify" \rangle Welcome to HTML $\langle /h_3 \rangle$

$\langle h_7$ align = "right" \rangle Welcome to HTML $\langle /h_7 \rangle$

2. Paragraph Tag (`<p>`)

- `<p>` tag marks a block of text as a paragraph.
- The web browser formats the text into a paragraph to fit the current page width.
- Breaking the document into paragraphs provides an easy way to format text.
- It is a container tag.

Eg:

```
<p> Here is the text for first  
paragraph of the document </p>
```

```
<p> Here is the text for second  
paragraph of the document </p>
```

```
<p align = "center"> Welcome </p>  
align can be left, right or center.
```

3. Line Break Tag (`
`)

- Anything following `
` tag starts from the next line. It is an empty element.
- Use `
` - for HTML
`
` - for XHTML [by space slash]

Eg: `<p> Welcome to
 Hyderabad </p>`
`Welcome to HTML
`
`a web design tool.`

4. Horizontal Lines. (<HR>)

- Horizontal lines are used to visually break-up sections of a document.
- The <hr> tag creates a line from the current position in the document to the right margin and breaks the line accordingly.
- It is an example of empty element. (Non Container)

Eg: <hr> <p> Welcome to HTML </p> ~~&~~
 <p> Welcome to web design <hr>

The attributes of <hr> tag are

align — Specifies the alignment of ~~the~~ <hr> element
 (The value can be left, center, right).

noshade — Specifies that <hr> element render in one solid color.

size — Specifies the height of <HR> element in pixels.

width — Specifies width of <hr> element in percentage of screen area.

Eg:

<hr> — HTML

<hr /> — XHTML

<hr align = "left" >

<hr size = 30 width = "40%" >

<hr size = 20 width = "80%" noshade >

5.

HTML formatting tags

(i) Bold Text (**)**

- Used to markup text as bold.
- The text to be bold is inserted between **(open tag) and** (closing tag).

Eg: **Welcome to HTML**

(ii) Italic Text (*)*

- Used to markup text as italic.
- The text appears within *...* is displayed in italicized.

Eg: *Welcome to HTML*

(iii) Underlined Text ()

- Used to markup text as underlined.
- The text appears in ... is displayed with underline.

Eg: Welcome to Hyd

(iv) strike Text (~~)~~

- Used to markup text as strike through.

Eg: ~~HTML is web development Tool~~

(v) Monospaced font (<tt>)

- Used to render/markup text as monospace font
- In monospace font all characters will present with same width.

Eg: <tt> welcome to HYD </tt>

(vi) Superscript text (<sup>)

- Used to markup text as superscript text
- The font size is same as the characters surrounding it, but displayed half a character's height above the other characters.

Eg: Hello ^{welcome to India}

The document created on ⁰³⁻⁰⁴⁻²⁰¹⁹

(vii) Subscript Text (<sub>)

- Used to render text as subscript text.
- The font size used is the same size as the characters surrounding it, but is displayed half a character's height beneath the other characters.

Eg: welcome to ₂₀₁₉

(viii) Larger Text (<big>)

- Used to markup text as big size
- The element displays one font size larger than the rest of text surrounding .

Eg: Welcome to <big> HYDERABAD </big>

(ix) Smaller Text (<small>)

- Used to markup text as small size
- The element displays one font size smaller than the rest of text surrounding .

Eg: Welcome to <small> Hyderabad </small>

(x) Grouping content (<div>)

- Used to group together several elements to create sections or subsections of a page.

Eg:

```
<div id = "101" align = "middle">
    <p> Welcome to Hyd </p>
    <b> Welcome to HTML </b>
</div>

<div id = "105" align = "right">
    <i> Welcome to India </i>
    <hr>
</div>
```

(xi) Grouping Content (``)

- Used to group inline elements only.
- If part of a sentence or paragraph to group together, `` element is used.

```
<p> Welcome to <span style="color:green"> Hyderabad  
</span> and  
<span style="color:red"> India </span>  
</p>
```

(x) Computer code (`<code>`)

- used to markup text as ~~marker~~ program code.
- Any programming code to appear on a web page should be placed inside `<code> ... </code>`
- Renders text as monospace font.

Eg: `<code> System.out.println </code>`

(xi) Strong Text (``)

- The important text to be displayed using ``
- It displays text in bold.

Eg: ` Welcome to Hyderabad `

HTML Attributes

Date _____
Page _____

- An attribute is used to define the characteristics of an HTML element.
- All attributes have name and value
 - name - The Property to set.
Eg: align, style, color etc
 - value - The value to be assigned to the property.
Eg: left, bold, red etc.
- W3C standardized to represent name & value of attribute in lower case.
- All attributes are case insensitive. (Not case sensitive)

1. Core Attributes

id - used to uniquely identify any element with in a HTML page.

Eg: `<p id="cse"> welcome to cse </p>`

`<p id="ece"> Welcome to ECE </p>`

title - to specify title for the element,
displays tool tip message

Eg: `<h3 title = "HTML"> Welcome to HTML </h3>`

class - Used to associate an element with a style sheet, and specifies the class of element.

style - Used to specify user defined styles for an element.

Eg: `<p style = "font-family : arial ; color : #FFDD22">`
welcome to HTML
`</p>`

2. Internationalization Attributes

dir - Indicates to the browser about the direction in which the text should flow.

The value can be ① ltr - left to right
② rtl - right to left

lang - Indicates the main language used in the document.

Eg: `<html lang = "en" dir = "rtl">`
=
`</html>`

3. Generic Attributes

Used with many of the HTML elements.

<u>Attribute</u>	<u>options/values</u>	<u>Function</u>
① align	right, left, center	Horizontal alignment of tags/text/images
② valign	top, middle, bottom	Vertically aligns tags Within an HTML element.
③ bgcolor	numeric, Hexadecimal, RGB values	Places a background color behind an element
④ background	URL	Places a background image behind element
⑤ width	Numeric value	Specify the width of tables, cells, images etc.
⑥ id	userdefined	uniquely identifies an element
⑦ class	User defined	To specify userdefined "Style classes"
⑧ title	User defined	To display tooltip (or) "pop-up" title of element

(xii) Emphasized Text (``)

Eg: `` Emphasized HTML Text ``

(xiii) Marked Text (`<mark>`)

— used to highlight or mark text.

Eg: `<h2>` Welcome to `<mark>` HTML `</mark>`
`</h2>`

(xiv) Deleted text (``)

Eg: This is the text `` removed ``

(xv) Inserted text (`<ins>`)

Eg: My favorite book is `<ins>` HTML `</ins>`

HTML Comments

Comment tags are used to specify the comments in source code, it can not be rendered on the webpage by the browser.

Single line → <!-- Write the comment text here -->

multiple lines → <!-- use multiple lines of comments here for documentation -->

HTML Colors

HTML colors are specified by using

① Predefined color names

Eg: Tomato, Orange, lightgray etc

HTML supports 140 standard color names.

Eg: welcome

② RGB value

A color can be specified as an RGB value.

rgb(red, green, blue)

red, green, blue → values defines the intensity from 0 to 255.

Eg: rgb(200, 150, 100).

rgb(0, 0, 0) → Black color

rgb(255, 255, 255) → White color

Ej:
 welcome to Hyderabad

(3) HEXadecimal values

A color can be specified with hexadecimal value by $\#rrggbb$

rr - red gg - green bb - blue	$\left. \begin{matrix} \\ \\ \end{matrix} \right\}$	Hexadecimal value from 00 to ff .
-------------------------------------	---	--------------------------------------

Ej:
 welcome to HTML

(4) HSL value

A color can be specified using hue (H), saturation (s) and lightness (L) as
 hsl (hue, saturation, lightness)

Hue - degree on the color wheel from 0 to 360.
 0 - red, 120 - green, 240 - blue

Saturation - percentage value.

0% → shade of gray

100% → full color

Lightness - percentage value.

0% → black /

100% is white

Eg:

welcome to HTML

 .

Lists

Date _____
Page _____

A List is a compact, aligned and justified related items together.

HTML provides 3 types of lists.

1. Unordered Lists
2. Ordered Lists
3. Definition lists.

1. Unordered List (``)

- It is a collection of related items that have no special order or sequence.
- `` tag is used to create unordered list
- `` tag (List Item) is used to add items to the list
- Each item in the list is marked with a bullet.

Eg:

```
<ul>
  <li> CSC </li>
  <li> ECE </li>
  <li> EEE </li>
</ul>
```

The attributes of `` tag are

type - to specify the type of bullet. The value can be

square - ■

disc - ● (By default)

circle - ○

title - To display pop-up or Tool tip message.

Ex: `<ul type="square" title="branches">`
 ` CSE `
 ` ECE `
 ` EEE `
 ``

2. Ordered List (``)

- It is a collection of related items that have a numeric or alphabetic sequence
- `` tag is used to create ordered list.
- `` tag is used to add each list item.
- Numbering starts at 1 and is incremented by 1 for each new list item.

Ex: ``
 ` CSE `
 ` ECE `
 ` EEE `
 ``

The attributes of `` tag are

Type - To specify the type of numbering.

The Value can be

"1" - Numeric order (By default)

"I" - Upper case Roman Numerals

"i" - Lower case Roman Numerals

"A" - Uppercase letters

"a" - Lowercase letters

title

— to display pop-up or Tool-Tip message.

start— to specify the starting point of numbering
(By default starts from one).**Eg:**

```
<OL type="I" start="3">
```

```
<LI> CSE </LI>
```

Q/p

```
<LI> ECE </LI>
```

↓

```
<LI> EEE </LI>
```

iii cse

iv ece

v eee

3. Definition List (<DL>)

- The definition list is a collection of list items, each list item consists of a term and its definition.
- It is used to present a glossary, list of terms or other name-value list.
- <DL> tag is used to create definition list.
- <DT> tag is used to create each term.
- <DD> tag is used to create each definition.

Eg:

```
<DL title="branches">
```

```
<DT> CSE </DT>
```

```
<DD> Computer Science Engineering </DD>
```

```
<DT> ECE </DT>
```

```
<DD> Electronics & Communication  
Engineering </DD>
```

```
</DL>
```

Creating a Nested List.

Eg:

```
<UL type = "square" title = "courses">
```

```
    <LI> UG courses
```

```
        <OL type = "1">
```

```
            <LI> CSE <LI>
```

```
            <LI> ECE <LI>
```

```
            <LI> EEE <LI>
```

```
        <OL>
```

```
    </LI>
```

```
    <LI> PG Courses
```

```
        <OL type = "1" start = "4">
```

```
            <LI> MBA <LI>
```

```
            <LI> MTech <LI>
```

```
        <OL>
```

```
    </LI>
```

```
</UL>
```

o/p]

o/p: ■ UG courses

1. CSE

2 ECE

3 EEE

■ PG courses

4. MBA

5 MTech

Working With Links

Date _____
Page _____

- Links or Hyperlinks are used to connect one webpage to another.
- The anchor tag (`<a>`) defines a hyperlink that links one page to another page.
- When we click a link, directed to webpage specified as the destination.
- `<a>` tag is used to create hyperlink.
- Anchor element also used to connect/link to different sections within the same webpage.

The attributes of `<a>` tag are

- ✓ **href** - Specifies the URL of the page the link goes to.
- name** - Specifies the name of an anchor, to specify the `section_name` within a page.
- target** - To specify where to open the linked document (`_blank, _parent, _self, _top, _framename`)

By default links will appear as

- An unvisited link is underlined & blue (link)
- A visited link is underlined & purple (vlink)
- An active link is underlined and red (alink)

Creating hyperlink -

```
<a href = "first.html" target = "_blank">
```

click here for HTML

O/P =>

click here for HTML

setting color to hyperlink

```
<body link="green" vlink="#AABBCc" alink="blue">  
  <a href="first.html" target="-blank">  
    click here  
  </a>
```

Linking to different sections within a page

```
<BODY>  
  <a href="#BOTTOM" > bottom </a>  
  <a name="TOP" > Top of Page </a>  
  == }  
  == }  
  == }  
  == }  
  == } Page Content
```

```
<a name="#BOTTOM" >  
  <H3> Bottom of Page </H3>  
  </a>  
  <a href="#TOP" > top </a>  
</BODY>
```

Images

Date _____
Page _____

- Image on a webpage is inserted using ``.
- `` tag defines a image
- Images are linked to the webpages and `` tag creates a holding space for the referenced image.
- closing tag is optional.

The attributes of `` element are

- align** - Alignment of image according to surrounding elements. (top, bottom, middle, left, right)
- alt** - to specify alternate text for an image
- border** - Width of border around an image in pixels.
- height** - specifies the height of an image in pixels.
- hspace** - horizontal spacing . i.e. left and right side of an image in pixels.
- width** - specifies width of image in pixels.
- vspace** - vertical spacing . i.e. top and bottom of image with space in pixels.
- src** - specifies the URL of an image.

Inserting an image

```
<img src = "c:\demo\first.jpg"> </img>
```

Displaying alternate text for image

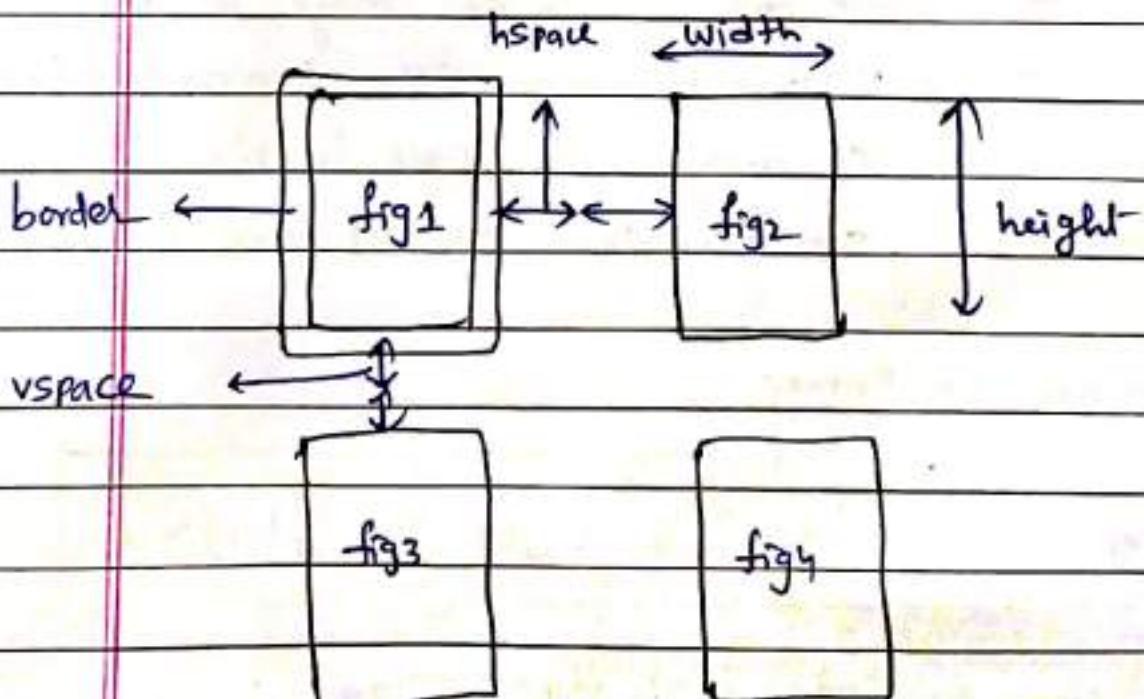
```
<img src = "c:\demo\first.jpg" alt = "my photo">
```

Adding border to an image & aligning image

```
<IMG src = "c:\demo\first.jpg"
      alt = "myphoto" height = 200 width = 200
      hspace = 5 vspace = 5 align = "right"
      border = 4 >
```

Using images as links

```
<a href = "first.html">
<IMG src = "c:\demo\fig.jpg" height = 20
      width = 40 alt = "mylink" >
</a>
```



HTML Tags Chart

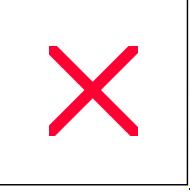
To use any of the following HTML tags, simply select the HTML code you'd like and copy and paste it into your web page.

Tag	Name	Code Example	Browser View
<!--	comment	<!--This can be viewed in the HTML part of a document-->	Nothing will show (Tip)
<a -	anchor	 Visit Our Site	Visit Our Site (Tip)
	bold	Example	Example
<big>	big (text)	<big>Example</big>	Example (Tip)
<body>	body of HTML document	<body>The content of your HTML page</body>	Contents of your web page (Tip)
 	line break	The contents of your page The contents of your page	The contents of your web page The contents of your web page
<center>	center	<center>This will center your contents</center>	This will center your contents
<dd>	definition description	<dl><dt>Definition Term</dt><dd> Definition of the term </dd><dt>Definition Term</dt><dd> Definition of the term </dd></dl>	Definition Term Definition of the term Definition Term Definition of the term
<dl>	definition list	<dl><dt>Definition Term</dt><dd>Definition of the term</dd><dt>Definition Term</dt><dd>Definition of the term</dd></dl>	Definition Term Definition of the term Definition Term Definition of the term
<dt>	definition term	<dl><dt> Definition Term </dt><dd>Definition of the term</dd><dt> Definition Term </dt><dd>Definition of the term</dd></dl>	Definition Term Definition of the term Definition Term Definition of the term
	emphasis	This is an Example of using the emphasis tag	This is an <i>Example</i> of using the emphasis tag
<embed>	embed object	<embed src="yourfile.mid" width="100%" height="60" align="center">	(Tip)
<embed>	embed object	<embed src="yourfile.mid" autoplay="true" hidden="false" loop="false"><noembed><bgsound src="yourfile.mid" loop="1"></noembed>	Music will begin playing when your page is loaded and will only play one time. A control panel will be displayed to enable your visitors to stop the music.
	font	Example	Example (Tip)

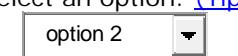
	font	Example	Example (Tip)
	font	Example	Example (Tip)
<form>	form	<pre><form action="mailto:you@yourdomain.com"> Name: <input name="Name" value="" size="10">
 Email: <input name="Email" value="" size="10">
 <center><input type="submit"></center> </form></pre>	<p>Name: <input type="text"/></p> <p>Email: <input type="text"/></p> <p><input type="button" value="Submit"/></p>
<h1> <h2> <h3> <h4> <h5> <h6>	heading 1 heading 2 heading 3 heading 4 heading 5 heading 6	<pre><h1>Heading 1 Example</h1> <h2>Heading 2 Example</h2> <h3>Heading 3 Example</h3> <h4>Heading 4 Example</h4> <h5>Heading 5 Example</h5> <h6>Heading 6 Example</h6></pre>	Heading 1 Heading 2 Heading 3 Heading 4 Heading 5 Heading 6
<head>	heading of HTML document	<head>Contains elements describing the document</head>	Nothing will show
<hr>	horizontal rule	<hr />	<p>Contents of your web page (Tip)</p> <hr/> <p>Contents of your web page</p>
<hr>	horizontal rule	<hr width="50%" size="3" />	<p>Contents of your web page</p> <hr/> <p>Contents of your web page</p>
<hr>	horizontal rule	<hr width="50%" size="3" noshade />	<p>Contents of your web page</p> <hr/> <p>Contents of your web page</p>
<hr> (Internet Explorer)	horizontal rule	<hr width="75%" color="#ff0000" size="4" />	<p>Contents of your web page</p> <hr style="background-color: red;"/> <p>Contents of your web page</p>
<hr> (Internet Explorer)	horizontal rule	<hr width="25%" color="#6699ff" size="6" />	<p>Contents of your web page</p> <hr style="background-color: blue;"/> <p>Contents of your web page</p>
<html>	hypertext markup language	<pre><html> <head> <meta> <title>Title of your web page</title> </head> <body>HTML web page contents </body> </html></pre>	Contents of your web page
<i>	italic	<i>Example</i>	Example
	image		 (Tip)

<input>	input field	<p>Example 1:</p> <pre><form method=post action="/cgi-bin/example.cgi"> <input type="text" size="10" maxlength="30"> <input type="Submit" value="Submit"> </form></pre>	<p>Example 1: (Tip)</p>
<input> (Internet Explorer)	input field	<p>Example 2:</p> <pre><form method=post action="/cgi-bin/example.cgi"> <input type="text" style="color: #ffffff; font-family: Verdana; font-weight: bold; font-size: 12px; background-color: #72a4d2;" size="10" maxlength="30"> <input type="Submit" value="Submit"> </form></pre>	<p>Example 2: (Tip)</p>
<input>	input field	<p>Example 3:</p> <pre><form method=post action="/cgi-bin/example.cgi"> <table border="0" cellspacing="0" cellpadding="2"><tr><td bgcolor="#8463ff"><input type="text" size="10" maxlength="30"></td><td bgcolor="#8463ff" valign="Middle"> <input type="image" name="submit" src="yourimage.gif"></td></tr> </table> </form></pre>	<p>Example 3: (Tip)</p>
<input>	input field	<p>Example 4:</p> <pre><form method=post action="/cgi-bin/example.cgi"> Enter Your Comments:
 <textarea wrap="virtual" name="Comments" rows=3 cols=20 maxlength=100></textarea>
 <input type="Submit" value="Submit"> <input type="Reset" value="Clear"> </form></pre>	<p>Example 4: (Tip)</p>
<input>	input field	<p>Example 5:</p> <pre><form method=post action="/cgi-bin/example.cgi"> <center> Select an option: <select> <option >option 1</option> <option selected>option 2</option> <option>option 3</option> <option>option 4</option> <option>option 5</option> <option>option 6</option> </select>
 <input type="Submit" value="Submit"></center> </form></pre>	<p>Example 5: (Tip)</p>
<input>	input field	Example 6:	Example 6: (Tip)

		<pre><form method="post" action="/cgi-bin/example.cgi"> Select an option:
 <input type="radio" name="option"> Option 1 <input type="radio" name="option" checked> Option 2 <input type="radio" name="option"> Option 3

 Select an option:
 <input type="checkbox" name="selection"> Selection 1 <input type="checkbox" name="selection" checked> Selection 2 <input type="checkbox" name="selection"> Selection 3 <input type="Submit" value="Submit"> </form></pre>	<p>Select an option:</p> <p><input type="radio"/> Option 1</p> <p><input type="radio"/> Option 2</p> <p><input type="radio"/> Option 3</p> <p>Select an option:</p> <p><input type="checkbox"/> Selection 1</p> <p><input checked="" type="checkbox"/> Selection 2</p> <p><input type="checkbox"/> Selection 3</p> <p><input type="button" value="Submit"/></p>
	list item	<p>Example 1:</p> <pre><menu> <li type="disc">List item 1 <li type="circle">List item 2 <li type="square">List item 3 </MENU></pre> <p>Example 2:</p> <pre><ol type="i"> List item 1 List item 2 List item 3 List item 4 </pre>	<p>Example 1: (Tip)</p> <ul style="list-style-type: none"> ● List item 1 ○ List item 2 ■ List item 3 <p>Example 2:</p> <ol style="list-style-type: none"> i. List item 1 ii. List item 2 iii. List item 3 iv. List item 4
<link>	link	<pre><head> <link rel="stylesheet" type="text/css" href="style.css" /> </head></pre>	
<marquee> (Internet Explorer)	scrolling text	<pre><marquee bgcolor="#cccccc" loop="-1" scrollamount="2" width="100%">Example Marquee</marquee></pre>	 <p>(Tip)</p>
<menu>	menu	<pre><menu> <li type="disc">List item 1 <li type="circle">List item 2 <li type="square">List item 3 </menu></pre>	<ul style="list-style-type: none"> ● List item 1 ○ List item 2 ■ List item 3
<meta>	meta	<pre><meta name="Description" content="Description of your site"> <meta name="keywords" content="keywords describing your site"></pre>	Nothing will show (Tip)
<meta>	meta	<pre><meta HTTP-EQUIV="Refresh" CONTENT="4; URL=http://www.yourdomain.com/"></pre>	Nothing will show (Tip)

<meta>	meta	<pre><meta http-equiv="Pragma" content="no-cache"></pre>	Nothing will show (Tip)
<meta>	meta	<pre><meta name="rating" content="General"></pre>	Nothing will show (Tip)
<meta>	meta	<pre><meta name="robots" content="all"></pre>	Nothing will show (Tip)
<meta>	meta	<pre><meta name="robots" content="noindex,follow"></pre>	Nothing will show (Tip)
	ordered list	<p>Numbered</p> <pre> List item 1 List item 2 List item 3 List item 4 </pre> <p>Numbered Special Start</p> <pre><ol start="5"> List item 1 List item 2 List item 3 List item 4 </pre> <p>Lowercase Letters</p> <pre><ol type="a"> List item 1 List item 2 List item 3 List item 4 </pre> <p>Capital Letters</p> <pre><ol type="A"> List item 1 List item 2 List item 3 List item 4 </pre> <p>Capital Letters Special Start</p> <pre><ol type="A" start="3"> List item 1 List item 2 List item 3 List item 4 </pre> <p>Lowercase Roman Numerals</p> <pre><ol type="i"> List item 1 List item 2 List item 3 List item 4 </pre>	<p>Numbered</p> <ol style="list-style-type: none"> List item 1 List item 2 List item 3 List item 4 <p>Numbered Special Start</p> <ol style="list-style-type: none"> List item 1 List item 2 List item 3 List item 4 <p>Lowercase Letters</p> <ol style="list-style-type: none"> List item 1 List item 2 List item 3 List item 4 <p>Capital Letters</p> <ol style="list-style-type: none"> List item 1 List item 2 List item 3 List item 4 <p>Capital Letters Special Start</p> <ol start="3" style="list-style-type: none"> List item 1 List item 2 List item 3 List item 4 <p>Lowercase Roman Numerals</p> <ol style="list-style-type: none"> List item 1 List item 2 List item 3 List item 4 <p>Capital Roman Numerals</p> <ol style="list-style-type: none"> List item 1 List item 2 List item 3 List item 4

		<p>Capital Roman Numerals</p> <pre><ol type="I"> List item 1 List item 2 List item 3 List item 4 </pre> <p>Capital Roman Numerals Special Start</p> <pre><ol type="I" start="7"> List item 1 List item 2 List item 3 List item 4 </pre>	<p>Capital Roman Numerals Special Start</p> <table> <tr> <td>VII.</td> <td>List item 1</td> </tr> <tr> <td>VIII.</td> <td>List item 2</td> </tr> <tr> <td>IX.</td> <td>List item 3</td> </tr> <tr> <td>X.</td> <td>List item 4</td> </tr> </table>	VII.	List item 1	VIII.	List item 2	IX.	List item 3	X.	List item 4
VII.	List item 1										
VIII.	List item 2										
IX.	List item 3										
X.	List item 4										
<option>	listbox option	<pre><form method=post action="/cgi-bin/example.cgi"> <center> Select an option: <select> <option>option 1</option> <option selected>option 2</option> <option>option 3</option> <option>option 4</option> <option>option 5</option> <option>option 6</option> </select>
 </center> </form></pre>	<p>Select an option: (Tip)</p> 								
<p>	paragraph	<p>This is an example displaying the use of the paragraph tag. <p> This will create a line break and a space between lines.</p> <p>Attributes:</p> <p>Example 1:
</p> <pre>
 <p align="left"> This is an example
 displaying the use
 of the paragraph tag.

 Example 2:
</pre> <p>Example 2:
</p> <pre>
 <p align="right"> This is an example
 displaying the use
 of the paragraph tag.

 Example 3:
</pre> <p>Example 3:
</p> <pre>
 <p align="center"> This is an example
 displaying the use
 of the paragraph tag.</pre>	<p>This is an example displaying the use of the paragraph tag.</p> <p>This will create a line break and a space between lines.</p> <p>Attributes:</p> <p>Example 1:</p> <p>This is an example displaying the use of the paragraph tag.</p> <p>Example 2:</p> <p>This is an example displaying the use of the paragraph tag.</p> <p>Example 3:</p> <p>This is an example displaying the use of the paragraph tag.</p>								
<small>	small (text)	<small>Example</small>	Example (Tip)								

<strike>	deleted text	< strike >Example</ strike >	Example									
	strong emphasis	< strong >Example</ strong >	Example									
<table>	table	<p>Example 1:</p> <pre><table border="4" cellpadding="2" cellspacing="2" width="100%"> <tr> <td>Column 1</td> <td>Column 2</td> </tr> </table></pre> <p>Example 2: (Internet Explorer)</p> <pre><table border="2" bordercolor="#336699" cellpadding="2" cellspacing="2" width="100%"> <tr> <td>Column 1</td> <td>Column 2</td> </tr> </table></pre> <p>Example 3:</p> <pre><table cellpadding="2" cellspacing="2" width="100%"> <tr> <td bgcolor="#cccccc">Column 1</td> <td bgcolor="#cccccc">Column 2</td> </tr> <tr> <td>Row 2</td> <td>Row 2</td> </tr> </table></pre>	<p>Example 1: (Tip)</p> <table border="1"> <tr> <td>Column 1</td> <td>Column 2</td> </tr> </table> <p>Example 2: (Tip)</p> <table border="1"> <tr> <td>Column 1</td> <td>Column 2</td> </tr> </table> <p>Example 3: (Tip)</p> <table border="1"> <tr> <td>Column 1</td> <td>Column 2</td> </tr> <tr> <td>Row 2</td> <td>Row 2</td> </tr> </table>	Column 1	Column 2	Column 1	Column 2	Column 1	Column 2	Row 2	Row 2	
Column 1	Column 2											
Column 1	Column 2											
Column 1	Column 2											
Row 2	Row 2											
<td>	table data	<table border="2" cellpadding="2" cellspacing="2" width="100%"> <tr> <td>Column 1</td> <td>Column 2</td> </tr> </table>	<table border="1"> <tr> <td>Column 1</td> <td>Column 2</td> </tr> </table>	Column 1	Column 2							
Column 1	Column 2											
<th>	table header	<pre><div align="center"> <table> <tr> <th>Column 1</th> <th>Column 2</th> <th>Column 3</th> </tr> <tr> <td>Row 2</td> <td>Row 2</td> <td>Row 2</td> </tr> <tr> <td>Row 3</td> <td>Row 3</td> <td>Row 3</td> </tr></pre>	<p>Colum n 1 Colum n 2 Colum n 3</p> <table> <tr> <td>Row 2</td> <td>Row 2</td> <td>Row 2</td> </tr> <tr> <td>Row 3</td> <td>Row 3</td> <td>Row 3</td> </tr> <tr> <td>Row 4</td> <td>Row 4</td> <td>Row 4</td> </tr> </table>	Row 2	Row 2	Row 2	Row 3	Row 3	Row 3	Row 4	Row 4	Row 4
Row 2	Row 2	Row 2										
Row 3	Row 3	Row 3										
Row 4	Row 4	Row 4										

		<pre></tr> <tr> <td>Row 4</td> <td>Row 4</td> <td>Row 4</td> </tr> </table> </div></pre>					
<title>	document title	<title>Title of your HTML page </title>	Title of your web page will be viewable in the title bar. (Tip)				
<tr>	table row	<table border="2" cellpadding="2" cellspacing="2" width="100%"> <tr> <td>Column 1</td> <td>Column 2</td> </tr> </table>	Column 1	Column 2	<table border="2" cellpadding="2" cellspacing="2" width="100%"> <tr> <td>Column 1</td> <td>Column 2</td> </tr> </table>	Column 1	Column 2
Column 1	Column 2						
Column 1	Column 2						
<tt>	teletype	<tt>Example</tt>	Example				
<u>	underline	<u>Example</u>	Example				
	unordered list	Example 1: List item 1 List item 2 Example 2: <ul type="disc"> List item 1 List item 2 <ul type="circle"> List item 3 List item 4 	Example 1: <ul style="list-style-type: none"> • List item 1 • List item 2 Example 2: <ul style="list-style-type: none"> • List item 1 • List item 2 <ul style="list-style-type: none"> ○ List item 3 ○ List item 4 				

Tables

Date _____
Page _____

- HTML table allows to arrange data such as text, images, links, forms and other tables.
- In a table data is arranged in the form of rows and cols of cells.
- The <TABLE> tag is used to create a table.
- Each row of the table is created using <TR> (Table row) tag.
- Headings of table columns are created/defined using <TH> tag (Table Heading).
- Each data value in the column is created using <TD> tag. (Table data)
- <caption> tag is used to associate a caption which provides short description about table.

<TABLE> Tag

- Used to create a table.
- The Elements in <table> tag are
<CAPTION>, <TR>, <TH>, <TD>
<THEAD>, <TBODY> & <TFOOT>

The Attributes of <table> are

align - horizontal alignment of the table in browser window. (left, center, right)

background - url of the background image for the table.

- bgcolor - background color of table cell
- border - to set borderwidth in pixels
- bordercolor - sets external border color
- cellpadding - spacing between cell walls and cell contents, in pixels
- cellspacing - distance between cells in pixels
- cols - no. of columns in the table
- datasrc - url or id of the data source object supplying data bound to this element.
- height - height of whole table in pixels
- width - width of table in pixels (or) Percentage of display area
- hspace - horizontal padding for whole table
- vspace - vertical padding for whole table
- title - holds additional information.
(tool tips display)
- frame - Outer border display.
(void, above, below, hsides, vsides, box, lhb, rhb, bse etc).

<TR> Tag

Creates a row in a table. It encloses <th> & <td> tags.

The attributes are

(left, right,
center)

align - horizontal alignment of text in each row

bgcolor - background color of row cells

bordercolor - sets the external border color for the row

style - inline styles indicating how to render element

title - holds additional information

valign - sets vertical alignment of data in
this row. (top, middle, bottom, baseline)

<TH> tag

Creates a table heading. Usually bold and
centered horizontally & vertically.

The Attributes are

align - horizontal alignment of content in tabel cell

bgcolor - Background ^{color} image for the table cell

background - background image for the table cell

bordercolor - Set external border color for the cell

colspan - Indicates how many cell columns of
the table this cell should span.

(by default 1).

rowspan - Indicates how many rows of the table
this cell should span

- height - Sets the height of the cell in pixels
- width - Specifies the width of the cell in pixels or percentage of screen area
- valign - Vertical alignment of the data in this cell.
- title - holds additional information.

<TD> Tag

To Specify data for a table cell.

Used inside <TR> element

The attributes are

align -

background -

bgcolor -

bordercolor -

colspan -

rowspan -

height -

width -

valign -

title -

<CAPTION> tag

To set caption for a table, specifies heading for table data.

The attributes are

align - horizontal alignment of the caption.
(left, center, right)

style - inline style indicating how to render the element.

title - holds additional information

valign - vertical alignment of caption
(top, middle, bottom, baseline)

<THEAD> Tag

Creates a table head when grouping rows

The attributes are

align - specifies horizontal alignment of text in the group (left, center, right)

bgcolor - background color for the group

style - inline styles how to render the element

title - holds additional information

valign - sets the vertical alignment of text
(top, middle, bottom, baseline).

<TBODY> Tag

Creates a table body when grouping rows.

The attributes are:

align - alignment of text in group.
(left, center, right, justify)

v-align - vertical alignment of text
(top, middle, bottom, baseline)

bgcolor - background color for the group

style - inline styles indicating how to render the element.

title - holds additional information.

<TFOOT> Tag

Creates a table foot when grouping rows.

The attributes are:

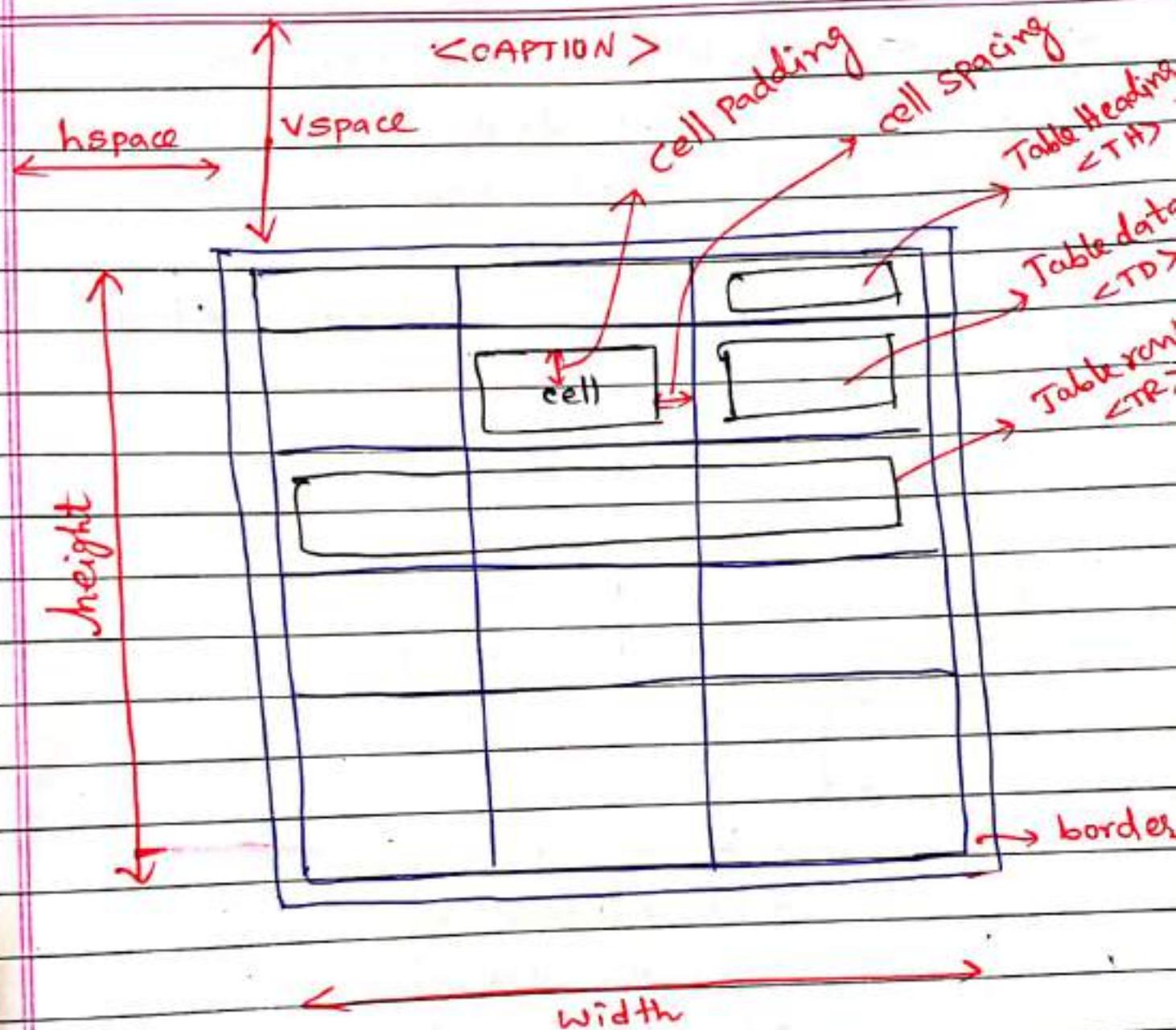
align - Alignment of text in the group

bgcolor - background color for the group

style - inline style, indicating how to render the element.

title - holds additional information

v-align - vertical alignment of text.



Eg ①

```
<table border = 3 bordercolor = "red"  
width = 400 height = 200  
cellspacing = 10 cellpadding = 15  
hspace = 50 vspace = 100 >  
<caption> Student Information </caption>
```

```
<tr>:
```

```
<th> Roll No </th>  
<th> Name </th>  
<th> Place </th>  
<th> Average </th>
```

```
</tr>
```

```
<tr>
```

```
<td> 501 </td>
```

```
<td> A. Ramesh </td>
```

```
<td> Hyderabad </td>
```

```
<td> 79.5 </td>
```

```
</tr>
```

```
<tr>
```

≡

```
</tr>
```

≡

```
</table>
```

Ex: ②

Colspan (Spanning columns)

colspan = 2

<table>

<tr>

<th colspan=2>

Roll NO &nbsp; &nbsp; Name </th>

<th> place </th>

<th> Average </th>

</tr>

<tr>

=

</tr>

</table>

Ex: ③

Rowspan (Spanning Rows)

<table>

<tr>

<th rowspan=4>

Roll NO

Name

Place

Average

</th>

...

<tr>

</tr>

Rowspan = 4

Frames

Date
Page

HTML frames are used to divide the web browser window into multiple sections, where each section can be loaded separately with a webpage.

- Frames allows to view multiple pages on same window at a time.
- The element <frameset> is used to hold the collection of frames. This element configures the frames on window/browser.
- A frameset is a collection of frames.
- <frame> tag is used to create frames.
 - ① <frameset> element actually takes the place of <body> element in a document that displays frames
 - ② <frameset> allows to organize both vertical & horizontal frames, or both.

< FRAMESET > Tag

It structures a document using frames.

uses <frame> element.

The attributes are

border - to set border thickness to all the frames within the frameset (0 to n)

bordercolor - sets the color of the borders for all frames in the frameset.

cols - sets the no. of columns in the frameset
(percentage of screen area. e.g: 25%, * → to show remaining space)

frameborder - Whether border is set to frame (0) or not. (yes or no)

framespacing - sets pixel spacing between frames.

rows - sets the no. of rows (frames) in the frameset. (1, 2, ..., n or *).

Style - inline style indicating how to render the element.

title - holds additional information for the element. (inline tool tips)

<Frame> tag

Used to create frames

The attributes are

bordercolor - Sets the color used for frame border

datafld - Name of the column of the data source object that supplies the bound data

datasrc - Gives the URL or ID of the data source object supplying data bound to this element.

frameborder - Sets whether or not borders surround the frame.

marginheight - Sets the size of top and bottom margins used in frame. (in pixels)

marginwidth - Sets the size of right and left margins used in the frame (in pixels)

name - Sets name of the frame

scrolling - Determines scrollbar action.
(auto, yes, no).

src - Specifies the url of the frame document

style - inline style indicating how to render the element

title - holds additional information
(tool tips).

Eg: ① Creating Vertical frames.

(i) $\langle \text{frameset} \text{ cols} = "25\%, 75\%" \rangle$
 $=$ || frames - 2
 $\langle / \text{frameset} \rangle$

(ii) $\langle \text{frameset} \text{ cols} = "25\%, * " \rangle$
 $=$ || frames - 2
 $\langle / \text{frameset} \rangle$

(iii) $\langle \text{frameset} \text{ cols} = "25\%, 52\%, * " \rangle$
 $=$ || 3 frames.
 $\langle / \text{frameset} \rangle$

② Creating horizontal frames.

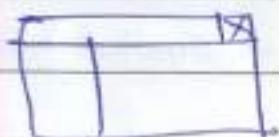
(i) $\langle \text{frameset} \text{ rows} = "30\%, 70\%" \rangle$
 $=$ || 2 frames
 $\langle / \text{frameset} \rangle$

(ii) $\langle \text{frameset} \text{ rows} = "30\%, 52\%, * " \rangle$
 $=$
 $\langle / \text{frameset} \rangle$

① Program to create Vertical frames

```
<html>
```

```
  <frameset cols="25%, *" bordercolor="red">
```



```
    <frame name="f1" src="index.html">
```

```
    <frame name="f2" src="content.html">
```

```
  </frameset>
```

```
</html>
```

② Program to create horizontal frames

```
<frameset rows="27%, * " >
```



```
    <frame name="f1" src="index.html">
```

```
    <frame name="f2" src="content.html">
```

```
  </frameset>
```

③ Program to create nested frames

```
<frameset cols="25%, 75%" >
```

```
  <frame src="index.html" name="f1">
```

```
  <frameset rows="20%, 71%, * ">
```

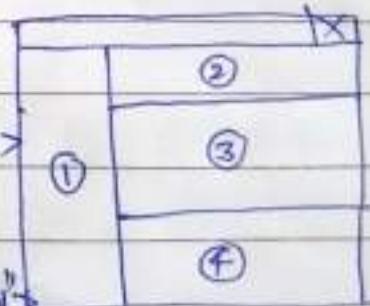
```
    <frame name="f2" src="head.html">
```

```
    <frame name="f3" src="content.html">
```

```
    <frame name="f4" src="copy.html">
```

```
  </frameset>
```

```
</frameset>
```



④ Creating vertical frames using Pixely :

```
<frameset cols = "200,*">
    <frame name = "f1" src = "head.html">
    <frame name = "f2" src = "content.html">
</frameset>
```

⑤ Program to create named frames.

~~<html>~~

<frameset name = "f1" src = "index.html">

<html>

<head>

<title> named frames </title>

</head>

<body>

<frameset cols = "30%,*,>

<frame name = "f1" src = "index.html">

<frame name = "f2" src = "Content.html">

</frameset>

</body>

</html>

index.html

```
<html>
```

```
  <body>
```

```
    <a href = "cse.html" target = "f2">cse</a>
```

```
    <br> <a href = "ece.html" target = "f2">ece</a>
```

```
  </body>
```

```
</html>
```

"if target = "newwindow" → displays the
webpage in a new window".

HTML Forms & HTML Controls

- Forms are used to handle html controls like buttons, textfields, text area etc.
- HTML forms are used to submit data to the server for processing.
i.e. A form packages all the data from a group of controls and all the data is sent to the server.
- Forms are not visible objects on the screen
- The `<Form>` tag is used to create forms
- A webpage can have multiple forms also.

<Form> Tag

Creates a HTML form, used to enclose HTML controls like buttons, textfields etc.

The attributes are

accept - list of content types that a server processing this form will handle correctly.

action - gives the URL that will handle form data in controls when submit button clicked.

autocomplete - if it is set to "true", automatically data in controls like text fields based on what the user has entered before.

class - class of the element

method - Indicates a ^{HTTP} method or protocol for sending data to the target action URL. (By default get). The values are get & post.

- name** - name to the form to reference in the script code
- style** - inline style indicating how to render the element.
- target** - indicates a named frame for the browser to display the form result in.
- title** - holds additional information (displayed in tool tips).

The various events on a form can be

onclick, onblur, ondrag, onkeydown,
onkeypress, onkeyup, onmousedown,
onmouseenter, onmouseleave, onsubmit, onreset etc.

Eg:
<body>

<form name = "form1" method = "post" >

=

User controls

=

</form>

</body>

=

forms contd...



- The purpose of HTML form is to allow the user to enter data on one end and send data to other end (client, server) through the webserver.
- Each form has a submit button that helps to send data through the server to the action URL.

HTML Controls

- Users interact with forms using HTML controls.
- HTML Controls are basic elements of a web page form.
- HTML Controls makes a webpage user friendly.
- Users enter data in webpage form with the help of HTML controls.

HTML Controls are placed / created using `<INPUT>` tag .

<input> Tag

Used to add HTML Controls, for sending input data.

The attributes are

Attribute	Value	Description
<td>left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom</td> <td>Alignment of text following the image. (used for <u>image control</u>)</td>	left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom	Alignment of text following the image. (used for <u>image control</u>)
alt →	any text	Alternate text for the image. used only with <u>image control</u> .
type →	button, checkbox file, hidden, image, password, radio, reset, submit text etc	type of the input element: The default value is " <u>text</u> ".
checked →	checked	Input element is checked when loads on webserver (used for <u>radio & checkbox</u>).
disabled →	disabled	Disables the input element when it loads on webserver. So that user cannot write or select from it.
maxlength →	number	Maximum no. of characters allowed in text field. (only with ' <u>text</u> ').

name →	field name	- Unique name for the element. Used with all controls.
readonly →	readonly	- Value of this field cannot be modified (used with <u>'text'</u>)
src →	URL	- URL of the image to display. (used with <u>'image'</u>)
value →	any text	- sets the caption

Adding a Button -

- Used to create a button in the HTML form.
- The type value is "button", in tag.

```
<input type = "button" value = "click here">
```

Adding a Submit Button -

- When submit button is clicked, all the data in the form is sent to the webserver.
- When user clicks, the data in the form is transferred to the URL specified in
- The type value is "submit" in tag

```
<input type = "submit" value = "Submit Here">
```

The attributes for submit button are

name	-	Name to the button
value	-	Text to be written on button
align	-	Defines the alignment of button
tabindex	-	Tab order of the button, used if there are multiple buttons in the webpage.

Adding a Reset Button

clear

- Reset button helps the user to clear all the data entered in text fields and keeps the form to the initial state and start all over again.
- When user clicks, the controls in the form are returned to their original state and values in fields are cleared.
- type value is "reset" in <input> tag

```
<input type = "reset" value = "clear">
```

Adding text field

I

- used to add a one line text in the form
- type value is "text" in <input> tag.

```
<input type = "text" name = "firstname"  
value = "Enter" size = 20 />
```

Adding a checkbox

- It is a small box with a check mark in it and user can either select or clear it by a click. Allows to Select more options.
- The type value is "checkbox" in <input> tag

CE : <input type = "checkbox" name = "branch" value = "ce" >

CSE : <input type = "checkbox" name = "branch" value = "cse" checked >

Adding a radio button

- It is displayed as a circle and a dot in the middle when selected.
- Radio buttons work in mutually exclusive groups and only one radio button can be selected at a time.
- The type value is "radio" in <input> tag.

Male : <input type = "radio" name = "gender" value = "yes" >

Female : <input type = "radio" name = "gender" value = "no" />

Adding a Text Area

- It is a multiple line text input control & displays the text entered in it.
- No. of rows and cols are specified using rows & cols attribute.
- The element `<Textarea>` is used to create it.

Text here

```
<textarea rows="10" cols="30">
```

With text if required initially

```
</textarea>
```

Adding a selection Control

- It is a drop down list used to select any of the options listed.
- `<select>` element is used to create it
- `<select>` used `<option>` and `<optgroup>` tag
- `<option>` to specify the item of list
- `<optgroup>` allows to group choices in the form

The courses are :

```
<select>
```

```
<optgroup label="UG courses">
```

```
<option value="cse"> CSE </option>
```

```
<option value="ece"> ECE </option>
```

```
</optgroup>
```

Date _____
Page _____

```
<optgroup label="PG courses">
    <option value="M Tech"> MTECH</option>
    <option value="MBA"> MBA </option>
</optgroup>
<select>
```

To select multiple options

(or) MULTIPLE

```
<select multiple>
```

=

```
<select>.
```

Adding a file control



- Used to create a file input for the form
- used to upload files.
- The type value is "file" in <input> tag.

```
<input type="file" name="filename"
       value="" size=30>
```

- The file control has two controls a text field and a browse button.
- When browse button clicked, the user can select a file from the disk.

Adding a hidden control

- It stores hidden data, data is not visible to users unless they view the page source.
- The type value is "hidden" in <input> tag.

```
<input type="hidden" name="hi"  
value = "welcome to forms">
```

Adding a image control

- It is similar to submit button.
- When user click on image, the form data sent to webserver
- The mouse coordinates are also passed to the form's action URL.
- The type value is "image" in <input> tag

```
<input type="image" src="filepath"  
name = "submit" value = "SUBMIT">
```

Adding password Control

- Creates a password text field, it masks the typed input.
- The type value is "password" in <input> tag

```
<input type="password" name="pwd" size=25>
```

The form data directed to the server by specifying the action URL and method, in form tag.

action - URL is the physical address of server to which the user data is redirected when submit button clicked.

method - The attribute has two values get & post.

① method = "get" → indicates that the form data (by default) has to be encoded by the web browser into a URL.

→ The form data visible in the address bar.

→ Used for small forms.

② method = "post" → indicates that the form data appears within message body.

- The form data invisible (hide)
- Used for large size forms.
- Send data as a packet

<form action = "demo.html" method = "post">

≡

</form>

Cascading Style sheets

(css)

Style
sheet

- Style sheets provide a way of customizing whole pages at once and in much richer details than the simple use of tags and attributes.
- Style sheets provide tag and attribute based style to improve look and feel of a webpage.
- Style sheets are implemented with the Cascading Style Sheet (css) specification.
- CSS is standardized by W3C, called ^{specifications} CSS specification.
- The CSS defines a rule with set of Properties.

Eg: H1 { font-size : 20pt } Rule

font-size → Property of css

20pt → value of css property.

- Style sheets are just a list of rules.
- The styles are created in 3 ways.
 - ① Embedded style sheets (Internal style sheets)
 - ② External style sheets
 - ③ Inline style sheets
- The style sheet implementation varies from browser to browser.

1. Inline styles

- "The "style" attribute of the every html tag is used to define the styles."
- used to apply style for a particular portion or to a particular element.

Ex: setting color of table cells.

```
<table border = 3 >
```

```
<TR>
```

```
<th style = "background-color : #aabbcc" >
```

S.NO </th>

inline style
attribute &
value

```
<th style = "background-color : #112233" > Name </th>
```

```
<th style = "background-color : #BBCCDD" > Place </th>
```

```
<TR>
```

```
<TR>
```

```
<TD style = "color : green ; font-style : italic" >
```

10/11 </TD>

```
<TD style = "color : red ; font-style : bold" >
```

A. Ramesh </TD>

```
<TD style = "color : blue ; font-style : italic" >
```

Hyderabad </TD>

```
<TR>
```

≡

```
</table>
```

"Multiple styles are

separated by ';' "

2. Embedded styles / Internal style sheets

" Using inline style, if several parts of the page adhere to the same style, All styles must be specified repeatedly".

- If same styles are repeated in multiple parts of the webpage Embedded styles are used.

- Embedded style sheets are ~~part~~ for a webpage.

" The styles applied throughout the webpage are collected and placed in one place".

- Embedded styles are specified in <style> tag of <head> element.

Eg:

```

<html>
  <head>
    <title> embedded styles </title>
    <style type = "text/css">
      BODY { background : #AABBCCC ; color: green }
      A:link { color : red }
      P { font-style : italic }
    </style>
  </head>
  <body>
    Welcome to Hyderabad
    <a href = "demo.html" > click here </a>
    <body> <p> India is rich in producing ... </p>
  </html>

```

for tag ←
for particular attribute ←

3. External style sheets

" If several webpages adhere to the same styles, the styles must be repeated for every page using embedded styles".

- If all pages in the website adhere to the same style external style sheets are used
- External styles are for whole website
- All the styles are stored separately in a file using file extension ".css".
- Useful when all the pages adhere to same style.

Eg:

↓ "demo.css"

```
BODY { background-color : #FFFFCC ;  
      font-family : Arial }  
  
A :link { color : #00AAFF }  
A :visited { color : #AABBCC }  
P { font-style : italic }
```

The <LINK> element in <head> tag is used to link css file for a webpage
<link relation = "stylesheet" href = "file.css" >

"first.html"

<html>

<head>

<link href = "demo.css" rel = "stylesheet">

</head>

<body>

Welcome to Hyderabad

 click here

<p> Hyd is for --- </p>

</body>

</html>

"sec.html"

<html>

<head>

<link href = "demo.css" rel = "stylesheet">

</head>

<body>

Welcome to Second bad

 click here

<p> SEC is for --- </p>

</body>

</html>

Style classes

- Allows to create styles in the form of style classes in external or embedded style sheets.
- To apply a style defined in a style class to an HTML element, assign class attribute of the HTML element to the name of the style class.
- The two types of style classes are
 - ① A universal style class starts with a dot operator (.) followed by the class name.
Ex: `<style> / / / / classname </style>`
`<style>`
class name { class definition
`</style>`
 - ② An element specific style class starts with the element name followed by a dot operator, which is followed by the class name.

`<style>`
ElementName . class name { class definition
`</style>`

Ex:

```
<html>
  <head>
    <title> style classes </title>
    <style type = "text/css">
      body { background-color : #f0f0ff }
      th { color { background-color : #800000 }
        green { background-color : #008000 }
      }
    </style>
  </head>
  <body>
    <table border = 2 >
      <caption> Student Data </caption>
      <tr>
        <th class = "color" > Name </th>
        <th class = "color" > Address </th>
      </tr>
      <tr>
        <td class = "green" > A - Ramu </td>
        <td class = "green" > Hyd </td>
      </tr>
      <tr>
        <td class = "green" > A - Suresh </td>
        <td class = "green" > Warangal </td>
      </tr>
    </table>
  </body>
</html>
```

Multiple Styles

```
<html>
```

```
  <head>
```

```
    <link rel = "stylesheet" href = "one.css" >
```

```
    <link rel = "stylesheet" href = "two.css" >
```

```
    <link rel = "stylesheet" href = "three.css" >
```

```
  </head>
```

```
  <body>
```

=

=

```
  </body>
```

```
</html>
```

- The use of several external stylesheets result in cascading the styles, which is a combination of styles for various HTML elements.
- * - If multiple files styles affect the same element, only the last one is used.

<marquee> Tag

The HTML <marquee> tag is used for scrolling piece of text or image displayed either horizontally across or vertically down.

The various specific attributes are

- behavior - Defines the type of scrolling (scroll, slide, alternate)
- bgcolor - defines the color of text
- direction - direction of scrolling the content (up, down, left, right)
- height - height of marquee (pixels or %)
- width - width of marquee (pixels or %)
- hspace - horizontal space around marquee (pixels)
- vspace - vertical space around marquee (pixels)
- loop - The default value is INFINITE, marquee loops endlessly. (or any no.)
- scrolldelay - Defines how long to delay between each jump (in seconds)
- scrollamount - Defines how far to jump (number)

Ex. <marquee direction = "up" height = 10

width = 300 loop = 10 scrolldelay = 3 >

Welcome to CSS

<marquee>

<pre> tag

- HTML <pre> tag is a block element, used to designate preformatted text.
- The text between <pre> tags has both its spaces and line break preserved and displayed in a fixed width font.

The Attributes are

width - Designates maximum number of characters per line

Eg:

```
<pre>
    welcome to HYD
    India is with
    Intellectual people
</pre>
```

CSS ATTRIBUTES LIST

S.No	Purpose	Property	Value os Property
1	Backgrounds	background-color	Any color value
		background-image	<code>background-image: url("bgdesert.jpg")</code>
		background-repeat	<code>repeat-x , repeat-y , no-repeat</code>
		background-attachment	<code>Fixed,scroll (image will not move while scrolling)</code>
		background-position	<code>right , top , bottom, left, center</code>
2	Borders	border-style	<code>Dotted, dashed, solid, double, groove, ridge, inset, outset, none, hidden</code>
		border-width	<code>5px</code>
		border-color	Any color value
		border-radius	<code>5px</code>
		border-top-style border-right-style border-bottom-style border-left-style	<code>Dotted, dashed, solid, double, groove, ridge, inset, outset, none, hidden</code>
		<code>p { border: 5px solid red;}</code>	
3	text	color	Any color value
		text-align	Center, left, right, justify
		text-decoration	<code>None, overline, line-through, underline</code>

		text-transform	Uppercase, lowercase, capitalize
		text-indent	50px (starting line gap)
		letter-spacing	3px, -3px
		line-height	0.8, 1.8 (Space between lines)
		direction	rtl,ltr
		word-spacing	10px
		text-shadow	3px 2px red (all three)
4	font	font-family	"Times New Roman", Times, serif
		font-style	Normal, italic, oblique
		font-size	14px or 1.875em or 100% ,10vw (eg: em=30px/16)
		font-weight	Normal, bold
		font-variant	Normal, small-caps
5	CSS Layout Overflow (used if content is more to present , it hide,scroll,visible etc)	overflow	visible, hidden, scroll, auto
		overflow-x	visible, hidden, scroll, auto
		overflow-y	visible, hidden, scroll, auto Eg: <DIV style="overflow:scroll" > ----take large text--- </DIV>
6	CSS Layout float	float	Right, left, none, inherit eg: img { float: right }
7	Margins	margin-top	100px
		margin-bottom	25px

		margin-right	35px
		margin-left	120px
		margin	Eg:margin: 25px 50px 75px 100px; (top, right, bottom and left)
8	Height and Width	height	100px
		width	500px
			<pre>div { height: 100px; width: 500px; background-color: powderblue; }</pre>
9	Outline	outline-style	Dotted, dashed, solid, double, groove, ridge, inset, outset, none, hidden
		outline-color	Any color value
		outline-width	Thin, thick, medium, any size in px, pt, cm, em eg: 8px
			-> shorthand property Eg: {outline: thick ridge pink;}
10	Links	color	Any color value
		a :link - a normal, unvisited link	Any color value
		a :visited - a link the user has visited	Any color value
		a :hover - a link when the user mouses over it	Any color value
		a :active - a link the moment it is clicked	Any color value

11	Color	background-color	Any color value
		Color (color of text)	Any color value
12	Box Model	Width Border Padding margin	<pre>div { width: 300px; border: 15px solid green; padding: 50px; //clears all sides with space margin: 20px; }</pre>
13	Links	list-style-type	Circle, square, upper-roman, lower-alpha
		list-style-image	list-style-image: url('sqpurple.gif');
		list-style-position	Outside, inside
14	Img	border-radius border padding width max-width: 100%; height: auto	<p>8px , 50%</p> <pre>img { border: 1px solid #ddd; border-radius: 4px; padding: 5px; width: 150px; } //displays images in thumb nails</pre>

HTML Entities

HTML Entities are used

Reserved characters in HTML must be replaced with character entities. (Eg: <, >, & etc)

Characters that are not present on your keyboard can also be replaced by entities. (pound , copyright).

Character entities are used to display reserved characters in HTML.

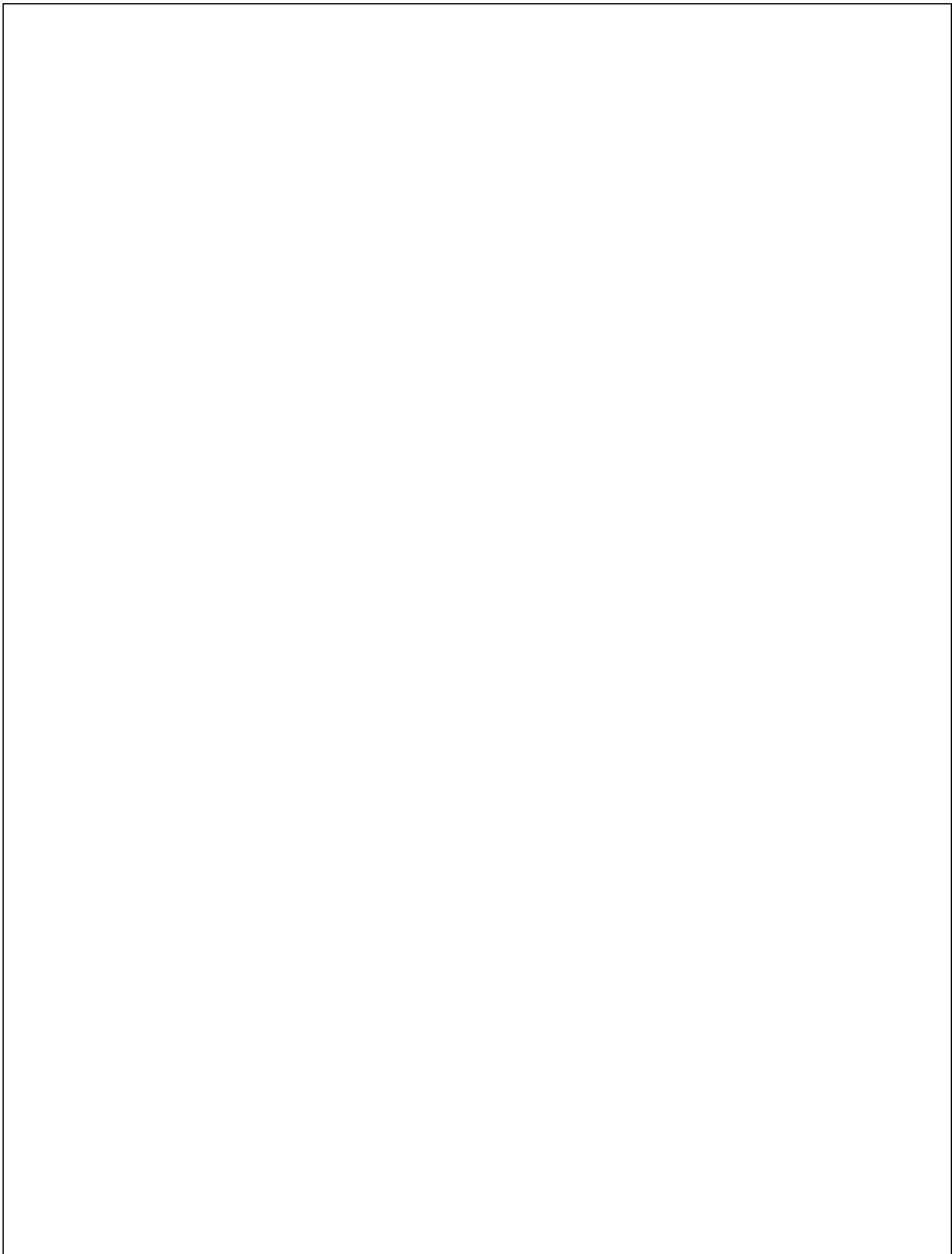
A character entity looks like this:

`&entity_name;`

OR

`&#entity_number;`

Result	Description	Entity Name	Entity Number
	non-breaking space	 	
<	less than	<	<
>	greater than	>	>
&	ampersand	&	&
"	double quotation mark	"	"
'	single quotation mark (apostrophe)	'	'
¢	cent	¢	¢
£	pound	£	£
¥	yen	¥	¥
€	euro	€	€
©	copyright	©	©
®	registered trademark	®	®



Java Script

Introduction to Javascript

Date _____
Page _____

DHTML (Dynamic HTML) provides different technologies required to make webpages dynamic and HTML.

DHTML is a combination of HTML, CSS & Scripting languages (JavaScript) and DOM.

JavaScript - Used to control, access and manipulate HTML elements.

CSS - Allows to control style & layout of webpages

DOM - Document object Model, defines a standard set of objects for HTML.

JavaScript

- A script is a program code written using a Scripting language.
- A Scripting language is a kind of programming language with less functionality.
Eg: JavaScript, VBScript, ASP, PHP
- JavaScript is an interpreted, client-side and Object based Scripting language that offers to create dynamic and interactive webpages.
- HTML & JavaScript applications can be developed on Microsoft Frontpage, Macromedia Dreamweaver, Macromedia HomeSite 5.

Javascript Characteristics

- ① Javascript is standardized by "European Computer Manufacturers Association" (EcMA), also called ECMA Script
- ② Originally named "Livescript", developed by netscape & Sun Microsystem
- ③ It is a Object Based Scripting language.
(No inheritance)
- ④ It is a Client-side Scripting language, the Scripts run in the browser.
- ⑤ The Script is processed by line by line, hence it is interpreted language
- ⑥ The Script engine (interpreter) is a built-in Component of the Web browser.
- ⑦ It is a case sensitive language
- ⑧ Uses Control statements, events and In-built objects to make webpages dynamic
- ⑨ Used to develop interactive and dynamic webpages.
- ⑩ It is relatively simple to learn & implement.
- ⑪ It reduces demand on server as it is a Client-side scripting language.
- ⑫ Variables need not to be declared before their use

- (B) Checking the compatibility of type can be done dynamically.
- (14) Java script objects are dynamic. i.e. Allows to change data and methods of an object during execution.
- (15) Javascript is used to validate the data on the webpage before submitting it to the server.
i.e. "Client-side Validation".
- (16) Javascript is an untyped language, i.e. a variable can hold any type of value.
- (16) Javascript is a free formed language, i.e. no constraints on spacing, line breaks etc.

Limitations

- ① Javascript does not allow reading or writing of file.
(for security reason)
- ② Not support to develop network applications
- ③ Doesn't have any multithreading or multiprocessor capabilities.

Features of Javascript

- ① All browsers support Javascripting, no need to use any plug-in.
- ② Uses structured programming language syntax.
(Similar to c/c++)
- ③ No need to place semicolon (;) at the end of statement.
- ④ Data type is bound to the value and not to the variable. Hence it is untyped language
- ⑤ Using "eval" function, the expression can be evaluated at runtime.
- ⑥ Provides built-in objects.
- ⑦ Supports use of regular expressions using which text-pattern matching is done.
used to validate webpage data.
- ⑧ It is a function programming. i.e.
Supports functions.
- ⑨ Light weight programming language for network-centric applications.
- ⑩ Less server interaction. i.e. validates user input before sending the page to server.
- ⑪ Immediate feedback to the user
- ⑫ Provides rich set of interfaces like popup windows, drag & drop components.
- ⑬ Effective event handling mechanism.

Embedding Javascript in HTML page

- The script is embedded in HTML document using `<SCRIPT>` tag.
- The browser processes the script code in `<script>` tag.
- The attributes of `<script>` tag are
 - type** - The type can be `"text/javascript"`, (or)
`"text/vbscript"` or `"text/ecmascript"`
 - language** - The attribute can have value `"Javascript"` or `"Vbscript"`.
 - src** - URL of the script file for embedding.
Used for external script files.

The script can be

① Inline script | embedded script

The script code is specified in the HTML document using `<SCRIPT>` tag in `<HEAD>`] `<BODY>`

```
<html>
```

```
<head>
```

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

```
language = "Javascript">
```

in
`<head>`
(or)

`<body>` tags

document.write ("welcome to Javascript")

```
</script>
```

```
</head>
```

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

```
<html>
```

② External script

The Java script code is stored in a separate file using ".js" extension.

Eg:

script.js

```
document.write ("External script");
document.write ("  
");
document.write ("Welcome to Javascript");
```

The Script file can be accessed in a webpage using <script> tag.

i.e.

```
<html>
  <head>
    <script src = "script.js" type = "text/javascript">
      </script>
    </head>
    <body>
      <h1> JavaScript </h1>
    </body>
  </html>
```

Reserved Words / Keywords

break continue delete for in
case default else function instanceof
catch do finally if new
return switch this throw try
typeof var void while with

JavaScript Comments

① Single line comments

// comment line appear here

② Multiline comments

/* comments in multiple lines will
appear here */

③ XHTML based comments

<!-- comments here -->

Variables in Javascript

- Variables are named locations in the memory and used to store data.

- A variable has name, value and memory address.

- "var" keyword is used to declare a variable.

- Javascript allows dynamic initialization. i.e.

- Variable can be declared when it is needed.

- Keywords cannot be used for variable names.

Eg: var no, name, avg;
var x;
Var sum = 0;

Eg:

- var x = 10; // integer type - x
-
- x = "Hello" // x is String type
-
- x = 123.45 // x is float type.

A variable in Javascript can hold any type of value. The particular type is known dynamically during last assignment.

"Hence Javascript" is untyped language

Control statements

① Conditional statements

if, if-else, nested if, else-if ladder,
switch. (case constant can be 'String' also).

② Loop / Iterative statements

while, do while, for

③ Jump statements

break, continue

Operators in Javascript

Arithmetic operators	-	$+, -, *, /, \cdot\%$, $++, --$
Assignment operators	-	$=, +=, -=, *=, /=, \cdot\% =$
Comparison operators	-	$==, !=, >, >=, <, <=$
Logical operators	-	$\&&, , !$
Conditional operators	-	$? :$ $x = (a < b) ? a : b ;$
Special operators	-	type of

The Precedence of operators is

$!, -, ++, --, \text{type of}$

Highest

$*, /, \cdot\%$

$+, -$ (subtraction)

$<, <=, >, >=$

$==, !=$

$\&&, ||, ?:$

$=, +=, -=, *=, /=, \cdot\% =$

Lowest

- ✓ The scope of variables can be local to the block, function and global.

Arrays in Java Script

- An array is a named collection of different values. The individual values are represented by their respective array elements, each of which has a unique index.
- In Java script, arrays are instances of the "Array" object.
- Arrays are dynamic, defined or created using "new" keyword.

**

In Javascript the values in an array need not to be of same data type.

Eg: A array can have both integer and string values.

```
Var arr-name = new Array( size );
Var arr-name = new Array();
```

Eg: Var a = new Array(10);

Var list = new Array();

An array can be initialized as

```
Var marks = new Array( 30, 40, 50, 60 );
```

```
Var data = new Array( 530, "A.Ramulu", 8.9 );
```

A particular element of an array can be accessed by specifying index. (starts with 0).

Eg. data[2] → 8.9.

In Javascript array a place holder can be specified to add/insert element later.

Eg: Var marks = new Array(10, 20, 30, 40, 50);
Later marks[2] = 80;
=

Functions in Javascript

A function refers to a cohesive block of statements that has a name and can be accessed or called from anywhere and any number of times in the script.

The function can be defined as

```
function fun_name (Par1, Par2 ... ParN)  
{  
    // function statements  
    [return expr]  
}
```

- A function can have zero or more parameters.
- A function may or may not return a value.

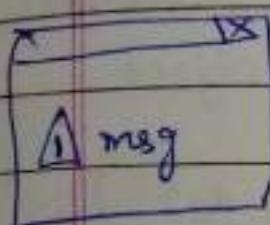
```
Eg: function add ( var a, b, c )  
{  
    var avg = (a+b+c)/3;  
    return avg;  
}
```

The function can be called as

```
Var y = add (10, 20, 30);
```

Built in functions of Javascript

- ① alert() -- Used to display information a
popup box | message box. (dialog)
/ used to display error messages once
a form is validated.
/ defined in "window" object.

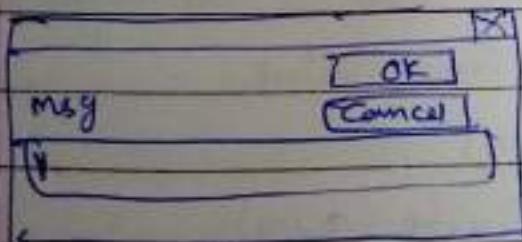


window.alert("message");
(or)

alert("Message");

Eg: window.alert("Enter a valid phone no");

- ② prompt() - Displays a message box containing a
text box with OK & Cancel buttons



/ It returns a text string when
OK is clicked and null
when Cancel is clicked.

/ used to accept input data
from the user for script code.

/ defined in "window" object

window.Prompt("message", "default value");
(or)

window.prompt("message");

Eg: var no = window.prompt("Enter a no", "0");
var no = prompt("Enter a no");

- ③ Confirm() - Displays a dialog box with two buttons OK & cancel.
When OK is clicked, returns true otherwise returns false.
- / Used to take user confirmation
, defined in window object

Window. Confirm ("would you like to do?");
(or)
Confirm ("Message here");

Eg: window. confirm ("Save or not ?");

- ④ isFinite() - , checks whether a value is finite or not.
- / Returns true if the argument is a finite value.

var isFinite (arg)

Eg. var x = isFinite (204);
document. writeln (x); // true .

- ⑤ isNaN() - Checks whether a number is ^{value is} or not.

(NaN) → Not a number .

" Returns true if the argument is string ".
isNaN (arg);

Eg: var x = isNaN("welcome"); // true
 var y = isNaN(2019); // false.

⑥ parseInt() - It parses the String and returns the first integer value found in the String.

parseInt(String);

Eg: var x = parseInt("2019"); // 2019.
 var y = parseInt("welcome"); // NaN
 "Returns NaN if it is not a no".

⑦ parseFloat() - parses the string and returns the first ^{float} ~~integer~~ value that found in the string.

parseFloat(String);

Eg: var x = parseFloat("32.64"); // 32.64
 var y = parseFloat("weldone"); // NaN.
 "Returns ~~too~~ NaN if it is not a no".

⑧ eval() - It accepts a string containing Javascript code, evaluates the argument and returning the resulting value.

eval(String);

Eg: eval("x = 50; y = 10; document.write(x+y)");
 ↳ Gives $\Rightarrow 60$.
 document.writeln(eval("10 * 5")); // 50

```
var x = 50;  
document.writeln(eval(x/z));
```

```
Var x = eval("104/3");  
= = = =
```

- ⑨ Number() - Used to convert the value of an object into a number.

Eg: var x = Number(Boolean(true)); // 1
var y = Number(String("1998")); // 1998
var z = Number(String("welcome")); // NaN

Primitive Data types

- ① Integer | Number
- ② Float
- ③ String
- ④ Boolean
- ⑤ Null
- ⑥ Undefined.

Programs on JavaScript Control Statements , Arrays and Functions

<!-- Greatest of 2 numbers -->

```
<!doctype html>
```

```
<html lang="en">
```

```
<head>
```

```
    <title>if-else statement</title>
```

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

```
        var a = window.prompt("Enter a number", "0");
```

```
        a = parseInt(a);
```

```
        var b= parseInt(prompt("Enter b number"));
```

```
        if(a > b)
```

```
            window.alert("The greatest is " +a);
```

```
        else
```

```
            alert("The greatest is " +b);
```

```
</script>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

<!-- Arranging 3 numbers in order -->

```
<!doctype html>
<html lang="en">
<head>
    <title>nested if statement</title>

</head>
<body>
    <script language="javascript" type="text/javascript" >
        var a = parseInt(window.prompt("Enter Marks Sub1"));
        var b= parseInt(prompt("Enter Marks Sub2"));
        var c= parseInt(prompt("Enter Marks Sub3"));
        var avg=(a+b+c)/3;

        if(avg >= 70)
            alert("First With Distinction");
        else if(avg >=60 && avg <70)
            alert("First Class");
        else if(avg >=50 && avg <60)
            alert("Second Class");
        else if(avg >=40 && avg <50)
            alert("Third Class");
        else
            alert("Fail");

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

<!-- Arranging 3 numbers in order -->

```
<!doctype html>
<html lang="en">
<head>
    <title>nested if statement</title>
    <script language="javascript" type="text/javascript" >
        var a = parseInt(window.prompt("Enter a number", "0"));
        var b= parseInt(prompt("Enter b number"));
        var c= parseInt(prompt("Enter c number"));
        if(a <b && a<c)
        {
            if(b < c)
                document.writeln("The order is " +a+ "," +b+"," +c);
            else
                document.writeln("The order is " +a+ "," +c+"," +b);
        }
        else if(b<a && b<c)
        {
            if(a <c)
                document.writeln("The order is " +b+ "," +a+"," +c);
            else
                document.writeln("The order is " +b+ "," +c+"," +a);
        }
        else
        {
            if(b <a)
                document.writeln("The order is " +c+ "," +b+"," +a);
            else
                document.writeln("The order is " +c+ "," +a+"," +b);
        }
    </script>
</head>
<body>

</body>
</html>
```

<!—Arithmetic operations using switch -->

```
<!doctype html>
<html lang="en">
    <title>Switch statement</title>
    </head>
    <body>
        <script language="javascript" type="text/javascript" >
            var a = parseInt(window.prompt("Enter a number"));
            var b= parseInt(prompt("Enter a number"));
            var c=0;
            var ch= prompt("Enter your choice");

            if(confirm("Would you like to Perform"))
            {
                switch(ch)
                {
                    //case constant can be string also
                    case "add" : c=a+b;alert("Result" +c);break;
                    case "sub" : c=a-b;alert("Result" +c);break;
                    case "mul" : c=a*b;alert("Result" +c);break;
                    case "div" : c=a/b;alert("Result" +c);break;
                    case "rem" : c=a%b;alert("Result" +c);break;
                    default: alert("Invalid operation");
                }
            }
            else
                alert("Invalid Data");

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

</body>
</html>
```

<!-- finding the grade using switch -->

```
<html>
  <head>
    <title> switch </title>
  </head>
  <script language="javascript" type="text/javascript">

    var n1 = parseInt(window.prompt("Enter the marks in Sub1"));
    var n2 = parseInt(window.prompt("Enter the marks in Sub2"));
    var n3 = parseInt(window.prompt("Enter the marks in Sub3"));
    var avg= (n1+n2+n3)/3;

    var ch = parseInt(avg/10); //parse to int

    switch(ch)
    {
      case 10:
      case 9:
      case 8:
      case 7: window.alert("Distinction");break;
      case 6: window.alert("First Class");break;
      case 5: window.alert("Second Class");break;
      case 4: window.alert("Third Class ");break;
      default: window.alert("Fail ");
    }

  </script>

  <body>
    </body>
  </html>

//switch statement Is fall-through
//switch allows string constant as case option
```

<!-- Check a number is Amstrong or Not. -->

```
<html>
  <head>
    <title> while demo </title>
  </head>
  <script language="javascript" type="text/javascript">

    var n = parseInt(window.prompt("Enter a Number"));
    var r , sum=0;
    var m =n;

    while(n > 0)
    {
      r= n % 10;
      sum= sum+(r*r*r);
      n=parseInt(n/10); //here n/10 float-> convert to int
    }
    document.writeln("The sum  is " +sum);
    if(sum == m)
      alert("The num is Amstrong" +m);
    else
      alert("The num is Not a Amstrong " +m);
  </script>

  <body>

  </body>
</html>
```

<!-- finding the Reverse of a number -->

```
<html>
  <head>
    <title> do-while demo </title>
  </head>
  <script language="javascript" type="text/javascript">

    var n = parseInt(window.prompt("Enter a Number"));
    var r , sum=0;

    do
    {
      r= n % 10;
      sum= sum * 10 +r;
      n=parseInt(n/10); //here n/10 float-> convert to int
    }while(n!=0);

    window.alert("The reverse num is " +sum);
  </script>

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

<!-- Prime number or NOt-->

```
<html>
  <head>
    <title> for loop </title>
  </head>
  <script language="javascript" type="text/javascript">
    var n = parseInt(window.prompt("Enter a Number"));
    var flag=true;

    for(var i=2 ; i <= parseInt(n/2); i++)
    {
      if (n % i == 0)
      {
        flag=false;
        break;
      }
    }

    if(flag==true)
      window.alert("The number is Prime "+n);
    else
      window.alert("The number is Not Prime "+n);

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

<!-- Multiplication Table -->

```
<html>
  <head>
    <title> for loop </title>
    <script language="javascript" type="text/javascript">
      var n = parseInt(window.prompt("Enter a Number"));
      for(var i=1; i<=10 ; i++)
      {
        document.write(n*i);
        document.writeln("<BR>");
      }
    </script>
  </head>
  <body>
    Multiplication Table  </body></html>
```

<!-- function to find factorial -->

```
<html>
  <head>
    <title> Functions </title>

    <script language="javascript" type="text/javascript">
      var n = parseInt(prompt("Enter a number"));
      var f = fact(n);
      document.writeln("The factorial is " +f);

      function fact( n )
      {
        var f =1;
        for(var i=1; i<=n ; i++)
          f= f*i;

        return f;
      }

    </script>

  </head>

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

<!-- array initializationa and creation -->

```
<html>
  <head>
    <title> Array Initialization and access </title>

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

      var data = new Array(530 , "A.Ramesh " , 9.8);

      for(var i=0;i<data.length ; i++)
        document.writeln(data[i] + " ");

      document.writeln("<BR>");

      var marks = new Array(87,81,77,88,76,92);

      for(var i=0;i<marks.length ; i++)
        document.writeln(marks[i] + " ");

      document.writeln("<BR>");
      var sum=0;
      for (var x in marks) //for - in loop
        sum=sum+marks[x];

      document.writeln("The average is " + (sum/6));

    </script>

  </head>

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

<!-- Sorting array elements -->

```
<html>
<head>
<title> Array Initialization and access </title>

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

    var n = parseInt(prompt("Enter the number of elements"));

    var a = new Array(n);

    window.alert("Enter the Array elements");

    for(var i=0 ; i < a.length ; i++)
        a[i] = parseInt(prompt("Enter the element" + i));

    for(var i=0; i < a.length ; i++)
    {
        for(var j=i+1 ; j < a.length; j++)
        {
            if(a[i] > a[j])
            {
                var temp ;
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }

    document.writeln("The elements after Sorting are <br>" );
    for(var i=0 ; i < a.length ; i++)
        document.writeln(a[i] + " " );

</script>
</head>
<body>

</body>
</html>
```

<!-- Two dimensional arrays Addition-->

```
<html>
<head>
<script language="JavaScript">
    var a = new Array(3);
    a[0] = new Array(3);
    a[1]= new Array(3);
    a[2] = new Array(3);

    var b = new Array(3);
    b[0] = new Array(3);
    b[1]= new Array(3);
    b[2] = new Array(3);

    var c = new Array(3);
    c[0] = new Array(3);
    c[1]= new Array(3);
    c[2] = new Array(3);

    for(var i=0; i< a.length ; i++)
        for(var j=0; j<a[i].length ; j++)
            a[i][j] = parseInt(prompt("Enter the elements of A matrix"));

    for(var i=0; i< b.length ; i++)
        for(var j=0; j<b[i].length ; j++)
            b[i][j] = parseInt(prompt("Enter the elements of B matrix"));

    for(var i=0; i< c.length ; i++)
        for(var j=0; j<c[i].length ; j++)
            c[i][j] = a[i][j] + b[i][j]

    for(var i=0; i< c.length ; i++)
    {
        for(var j=0; j<c[i].length ; j++)
            document.writeln(c[i][j] + "  &nbsp;"); 
            document.writeln("<BR>"); 
    }
</script>
<head>
<body>
</body>
</html>
```

JavaScript Event Handling

<!-- To access data from text feilds in Script onclick and onreset events -->

```
<html>
<head>
<script language="javascript" type="text/JavaScript" >

    function fun1()

    {

        var x = parseInt(document.f1.t1.value);

        var y = parseInt(document.f1.t2.value);

        var z = parseInt(document.f1.t3.value);

        var p = (x+y+z)/3;

        document.f1.t4.value = p;

    }

</script>
</head>
<body>

<form name="f1" onreset="confirm('Would you like to clear')" >

    <center>

        Marks in Sub1: <input type="text" name="t1" > <BR> <BR>

        Marks in Sub1: <input type="text" name="t2" > <BR> <BR>

        Marks in Sub1:<input type="text" name="t3" > <BR> <BR>

        Average is :<input type="text" name="t4" > <BR> <BR>

        <input type="reset" name="r1" value="clear" > &nbsp; &nbsp; &nbsp;

        <input type="button" name="b1" value="FindAvg" onclick="fun1()">

    </center>

</form>

</body></html>
```

```
<!-- To send data through arguments to JavaScript function
```

```
onclick and onreset -->
```

```
<html>
```

```
<head>
```

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

```
     function find(a , b ,c)  
     {           var x = parseInt(a);  
                 var y = parseInt(b);  
                 var z = parseInt(c);  
                 var p = (x+y+z)/3;  
                 document.f1.t4.value = p;  
     }
```

```
   </script>
```

```
 </head>
```

```
 <body>
```

```
   <form name="f1" onreset="confirm('Would you like to clear')" >
```

```
     <center>
```

```
       Marks in Sub1: <input type="text" name="t1" id="m1" > <BR> <BR>
```

```
       Marks in Sub1: <input type="text" name="t2" id="m2" > <BR> <BR>
```

```
       Marks in Sub1:<input type="text" name="t3" id="m3" > <BR> <BR>
```

```
       Average is :<input type="text" name="t4" id="res" > <BR> <BR>
```

```
       <input type="reset" name="r1" value="clear" > &nbsp; &nbsp; &nbsp;
```

```
     <b><input type="button" name="b1" value="FindAvg"  
           onclick="find(m1.value,m2.value,m3.value)">
```

```
     </center>
```

```
   </form>
```

```
 </body>
```

```
</html>
```

```

<!-- To access element by id onclick ,onload , onmouseup ,onmousedown, onmouseover .  

background color change  

document.body.style.backgroundImage = "url('filename')"  

document.body.style.background ="#110022" -->  

<html>  

<head>  

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

    function fun1()  

    {  

        document.body.style.backgroundImage = "url('Jellyfish.jpg')"  

    }  

    function fun2()  

    {  

        document.getElementById("uname").focus();  

    }  

</script>  

</head>  

<body onload="document.body.style.background ='#110022'"  

onmousedown="fun1()"  

onmouseup="document.body.style.background ='#EE1100'"  

onmouseover="fun2()">  

    <form name="f1" >  

        <input type= "text" name="t1" id="uname" size=50> <br>  

        <input type="button" name="b1" id="100" value="click here"  

        onclick="alert(' Hi ' + document.getElementById('uname').value)" ><br>  

    </form>  

</body>  

</html>

```

<!--accessing the checkbox status -->

```
<html>
<head>
<script language="javascript">

function fun1()

{
    var x = document.getElementById("ch1").value;
    alert("x value is " +x);
    //returns on if no value field in checkbox. if selected
    // returns the value field string of the checkbox

    if(x=="cse")
    {
        document.writeln(" cse selected");
    }
    else
    {
        document.writeln(" cse Not Selected");
    }
}

</script>
<body>

<form name="f1" >
    <input type="checkbox" value="cse" id="ch1" onclick="fun1()" > CSE
</form>
</body>
</html>
```

<!--Handling the Radio buttons -->

```
<!DOCTYPE html>

<html>
<body>
<script language="javascript" type="text/javascript">
function fun1()
{
    document.getElementById('male').checked = true;
    document.getElementById('fem').checked = false;
    document.getElementById("div1").innerHTML = "Gender Selected " +"MALE";
}
function fun2()
{
    document.getElementById('male').checked = false;
    document.getElementById('fem').checked = true;
    document.getElementById("div1").innerHTML = "Gender Selected " +"FEMALE";
}

</script>
<form>
    Gender :
    <input type="radio" name="gen" id="male" onclick="fun1()"> MALE &nbsp;&nbsp;
    <input type="radio" name="gen" id="fem" onclick="fun2()"> FEMALE
</form>
<div id="div1" ></div>
</body>
</html>
```

JavaScript Objects

- JavaScript is an **object –based** scripting language.
- An **object** is a programmable entity used in script.
- JavaScript defines several built in objects to use in different parts of the web page.

Eg: **Array, String Window, document etc.**

- Every object has properties and methods.
 - **Property** - is an attribute of a object.
eg: For Array object **length** is the property.
 - **Method** – action or task that can be performed on the object.
eg: **sort ()** is performed on Array object.
- Every HTML element in JavaScript is an Object.

The commonly used **Language** objects are

1. **Array**
2. **String**
3. **Math**
4. **Number**
5. **Boolean**
6. **Date**

Array Object

Array object allows creating and manipulating a series of values that are represented by a single name.

The array can be constructed using

1. Using array literal.

```
var data = [530 , "A.Ramesh" , 8.9];
```

2. Creating instance of Array object

```
var data= new Array();  
data[0]= 530;  
data[1] = "A.Ramesh";  
data[2] = 8.9
```

3. Using Constructor.

```
Var data = new Array(530 , "A.Ramesh" ,8.9);
```

The properties of Array object are

length: returns the number of elements in the array.

The methods of Array Object are

Method Name	Description	Example
concat()	It returns a new array object that contains two or more merged arrays. <code>array.concat(arr1,arr2,...,arrn)</code>	<pre>var a1=[20,30,40]; var a2=[60,70,80]; var result=a1.concat(a2); document.writeln(result);</pre>
join()	It combines all the elements of an array into a string and return a new string. We can use any type of separators to separate given array elements. <code>array.join(separator)</code>	<pre>var a=[530 , "A.Ramesh" , 8.9] var result=arr.join() document.write(result); // var result=arr.join('-')</pre>
reverse()	It reverses the elements of given array. <code>array.reverse()</code>	<pre>var a=["HYD","DELHI","CHENNAI"]; var rev=a.reverse(); document.writeln(rev);</pre>
sort()	It returns the element of the given array in a sorted order. <code>array.sort()</code>	<pre>var a=[12,14,11,18,5]; var result=arr.sort(); document.writeln(result);</pre>

<u>indexOf()</u>	<p>It searches the specified element in the given array and returns the index of the first match.</p> <p>array.indexOf(element,index)</p> <p>element - It represent the element to be searched.</p> <p>index - It represent the index position from where search starts. It is optional.</p>	<pre>var a=[12,32,22,31,32,58,28]; var result= a.indexOf(32); document.writeln(result); //1</pre> <p>Or</p> <pre>var a=[32,22,31,32,58,28]; var result= a.indexOf(32,2); document.writeln(result); //4</pre>
<u>lastIndexOf()</u>	<p>It searches the specified element in the given array and returns the index of the last match.</p> <p>array.lastIndexOf(element,index)</p> <p>element - It represent the element to be searched.</p> <p>index - It represent the index position from where search starts. It is optional.</p>	<pre>var a=[12,32,22,31,32,58,28]; var result= a.lastIndexOf(32); document.writeln(result);</pre> <p>Or</p> <pre>var a=[32,22,31,32,58,28]; var result= a.lastIndexOf (32,2); document.writeln(result);</pre>
<u>push()</u>	<p>It adds one or more elements to the end of an array.</p> <p>array.push(element1,element2....elementn)</p>	<pre>var a=["HYD","CHENNAI"]; a.push("DELHI"); document.writeln(a);</pre>
<u>pop()</u>	<p>It removes and returns the last element of an array.</p> <p>array.pop()</p>	<pre>var a=[20,34,56,78]; document.writeln(a.pop()); document.writeln(a);</pre>
<u>slice()</u>	<p>It returns a new array containing the copy of the part of the given array.</p> <p>array.slice(start,end)</p>	<pre>var a=[12,13,48,28,56,88,34] var result=a.slice(2,4); document.writeln(result);</pre>
<u>shift()</u>	<p>It removes and returns the first element of an array.</p> <p>array.shift()</p>	<pre>var a=[12,13,48,28,56,88,34] var result=a.shift(); document.writeln(result);</pre>
<u>unshift()</u>	<p>It adds one or more elements in the beginning of the given array.</p> <p>array.unshift(elem1,elem2,....,elemn)</p>	<pre>var a=[12,13,48,28,56,88,34] var result=a.unshift(89,99); document.writeln(result);</pre>

String Object

String object allows creating and manipulating a string of characters that are enclosed in single or double quotes.

The JavaScript **string** is an object that represents a sequence of characters.

The strings are created in two ways.

- Using string literal.

```
var stringname="string value"
```

```
var str="Welcome to JavaScript";
document.write(str);
```

- Using string object (Constructor)

```
var stringname=new String("string literal");
```

```
var str=new String("Welcome to JavaScript");
document.write(stringname);
```

The properties of String object are

length: returns the number of characters in the string.

The methods of String Object are

Method Name	Description	Example
<u>charAt()</u>	It provides the char value present at the specified index. string.charAt(index)	var str="JavaScript"; document.writeln(str.charAt(4));
<u>concat()</u>	It provides a combination of two or more strings. string.concat(str1,str2,...,strn)	var x="welcome"; var y="to"; var z=" India"; document.writeln(x.concat(y,z));
<u>indexOf()</u>	It returns the index position of char value passed with method. indexOf(ch)	var x="JavaScript"; document.write(x.indexOf('a'));
	It start searching the element from the provided index value and then returns the index position of specified char value. indexOf(ch,index)	var str="welcome javascript"; document.write(str.indexOf('e',3));

	<p>It returns the index position of first character of string passed with method. indexOf(str);</p>	<pre>var str=" webtech is to do web "; document.write(str.indexOf("web"));</pre>
	<p>It start searching the element from the provided index value and then returns the index position of first character of string. indexOf(str,index);</p>	<pre>var str=" webtech is to do web "; document.write(str.indexOf("web",2));</pre>
match()	<p>It searches a specified regular expression in a given string and returns that regular expression if a match occurs. string.match(regexp)</p>	<pre>var str="Javascript"; document.writeln(str.match("Java"));</pre>
replace()	<p>It replaces a given string with the specified replacement. string.replace(originalstr,newstr)</p>	<pre>var str="secbad"; document.writeln(str.replace("sec","hyd")); var str="secbad is famous for irani chai , seabad has clock tower"; document.writeln(str.replace(/sec/g,"hyd"));</pre>
search()	<p>It searches a specified regular expression in a given string and returns its position if a match occurs. string.search(regexp)</p>	<pre>var str="JavaScript is a scripting language. Scripting languages are often interpreted"; document.writeln(str.search("scripting")); document.writeln(str.search(/Scripting/)); document.writeln(str.search(/Scripting/i)); // i- ignore case sensitive</pre>
substring()	<p>It is used to fetch the part of the given string on the basis of the specified index. string.substring(start,end) ;</p>	<pre>var str="hyderabad"; document.writeln(str.substring(2,4)); var str="hyderabad "; document.writeln(str.substring(3));</pre>
toLowerCase()	<p>It converts the given string into lowercase letter. string.toLowerCase()</p>	<pre>var str = "weCOME to HYD"; document.writeln(str.toLowerCase());</pre>
toUpperCase()	<p>It converts the given string into uppercase letter. string.toUpperCase()</p>	<pre>var str = " weCOME to HYD "; document.writeln(str.toUpperCase());</pre>

<u>lastIndexOf()</u>	<p>It returns the last index position of char value passed with method.</p> <p>lastIndexOf(ch)</p>	<pre>var web="Learn JavaScript on Java script"; document.write(web.lastIndexOf('a'));</pre>
	<p>It starts searching the element from the provided index value in the inverse order and then returns the index position of specified char value.</p> <p>lastIndexOf(ch,index)</p>	<pre>var web="Learn JavaScript on Javascript" ; document.write(web.lastIndexOf('a',10));</pre>
	<p>It returns the index position of first character of string passed with method.</p> <p>lastIndexOf(str)</p>	<pre>var web="Learn JavaScript on Javascript" ; document.write(web.lastIndexOf("Java"));</pre>
	<p>It starts searching the element from the provided index value and then returns the index position of first character of string.</p> <p>lastIndexOf(str,index)</p>	<pre>var web="Learn JavaScript on Javascript" ; document.write(web.lastIndexOf("Java" ,19));</pre>
substr()	<p>It is used to fetch the part of the given string on the basis of the specified starting position and length.</p> <p>string.substr(start,length)</p>	<pre>var str="hyderabad"; document.writeln(str.substr(3,4)); var str="hyderabad"; document.writeln(str.substr(4));</pre>
slice()	<p>It is used to fetch the part of the given string. It allows us to assign positive as well negative index.</p> <p>string.slice(start,end)</p>	<pre>var str = "Javascript"; document.writeln(str.slice(2,5)); var str = "Javascript"; document.writeln(str.slice(5)); //from begining document.writeln(str.slice(-5)); //end of string</pre>
<u>toString()</u>	<p>It provides a string representing the particular object.</p> <p>object.toString()</p>	<pre>var str=new String("Javascript") document.writeln(str.toString());</pre>
<u>valueOf()</u>	<p>It provides the primitive value of string object.</p> <p>string.valueOf()</p>	<pre>var str=new String("Javascript"); document.writeln(str.valueOf());</pre>

Date Object

Date object allows to create and manipulate date and time.

The JavaScript date object can be used to get year, month and day. We can display a timer on the webpage with the help of JavaScript date object. Using different Date constructors to create date objects. It provides methods to get and set day, month, year, hour, minute and seconds.

The constructors of Date object are

1. Date()

```
var d = new Date();
```

2. Date(milliseconds)

```
var d = new Date(1542568925); // 01 January 1970 plus + msec
```

3. Date(dateString)

```
var d = new Date("2015-03-25");
var d = new Date("03/25/2015");

var d = new Date("Mar 25 2015");
var d = new Date("January 25 2015");
var d = new Date("October 13, 2014 11:13:00");
```

4. Date(year, month, day, hours, minutes, seconds, milliseconds)

```
var d = new Date(2018, 11, 24, 10, 33, 30, 0);

var d = new Date(2018, 11, 24, 10, 33, 30);
var d = new Date(2018, 11, 24, 10, 33);
var d = new Date(2018, 11, 24);
var d = new Date(2018);
```

The methods of Date Object are

Method Name	Description	Example
getDate()	It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time. dateObj.getDate()	var d=new Date(); document.writeln("Today's day: "+d.getDate()); var d=new Date("July 8, 2019 16:22:10"); document.writeln(d.getDate())
getDay()	It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time. dateObj.getDay()	var day=new Date(); document.writeln(day.getDay()); var day=new Date("July 8, 2019 16:22:10"); document.writeln(day.getDay())
getMonth()	It returns the integer value between 0 and 11 that represents the month on the basis of local time. dateObj.getMonth()	var date=new Date(); document.writeln(date.getMonth()+1); var date=new Date("July 8, 2019 16:22:10"); document.writeln(date.getMonth()+1)
getFullYear()	It returns the integer value that represents the year on the basis of local time. dateObj.getFullYear()	var year=new Date(); document.writeln(year.getFullYear()); var year=new Date("July 8, 2019 16:22:10"); document.writeln(year.getFullYear());
getHours()	It returns the integer value between 0 and 23 that represents the hours on the basis of local time. dateObj.getHours()	var hour=new Date(); document.writeln(hour.getHours()); var hour=new Date("July 8, 2019 16:22:10"); document.writeln(hour.getHours());
getMinutes()	It returns the integer value between 0 and 59 that represents the minutes on the basis of local time. dateObj.getMinutes()	var min=new Date(); document.writeln(min.getMinutes()); var min=new Date("July 8, 2019 16:22:10"); document.writeln(min.getMinutes());

getSeconds()	<p>It returns the integer value between 0 and 59 that represents the seconds on the basis of local time. // from 01 January 1970</p> <pre>dateObj.getSeconds()</pre>	<pre>var sec=new Date(); document.writeln(sec.getSeconds()); var sec=new Date("July 8, 2019 16:22:10"); document.writeln(sec.getSeconds());</pre>
getMilliseconds()	<p>It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.</p> <pre>dateObj.getMilliseconds()</pre>	<pre>var milli =new Date(); document.writeln(milli.getMilliseconds()); var milli =new Date("July 8, 2019 16:22:10"); document.writeln(milli.getMilliseconds());</pre>
 setDate()	<p>It sets the day value for the specified date on the basis of local time.(1 to 31)</p> <pre>setDate(dd);</pre>	<pre>var d = new Date(); d.setDate(15);</pre>
setDay()	<p>It sets the particular day of the week on the basis of local time. (0 to 6)</p> <pre>setDay(day)</pre>	<pre>var d = new Date(); d.setDay(3);</pre>
setFullYear()	<p>It sets the year value for the specified date on the basis of local time.</p> <pre>setFullYear(yyyy,mm,dd);</pre>	<pre>var d = new Date(); d.setFullYear(2020); var d = new Date(); d.setFullYear(2020, 11, 3);</pre>
setMonth()	<p>It sets the month value for the specified date on the basis of local time.(0-11)</p> <pre>setMonth(mm)</pre>	<pre>var d = new Date(); d.setMonth(11);</pre>
setHours()	<p>It sets the hour value for the specified date on the basis of local time.</p> <pre>setHours(hh , mm , ss);</pre>	<pre>var d = new Date(); d.setHours(22);</pre>
setMinutes()	<p>It sets the minute value for the specified date on the basis of local time.</p> <pre>setMinutes(mm,ss)</pre>	<pre>var d = new Date(); d.setMinutes(30);</pre>
setSeconds()	<p>It sets the second value for the specified date on the basis of local time.</p> <pre>setSeconds(ss)</pre>	<pre>var d = new Date(); d.setSeconds(30);</pre>

<code>setMillisecond()</code>	<p>It sets the millisecond value for the specified date on the basis of local time.</p> <pre>setMilliSeconds(msec); // 01 January 1970</pre>	<pre>var d = new Date(); d.setMilliSeconds(12563254825);</pre>
<code>toString()</code>	<p>It returns the date in the form of string.</p>	<pre>Var d = new Date(); var str = d.toString();</pre>
<code>valueOf()</code>	<p>It returns the primitive value of a Date object.</p>	<pre>var d = new Date(); var str = d.valueOf();</pre>
<p>Similarly all the methods with UTC time zone.(Coordinated Universal Time (UTC)) Eg: <code>setUTCDate()</code> , <code>getUTCMonth()</code> etc</p>		

Math Object

The JavaScript `Math` object provides several constants and methods to perform mathematical operation.

The `Math` object doesn't have constructors.

Method Name	Description	Example
<code>abs()</code>	<p>It returns the absolute value of the given number.</p> <pre>Math.abs(num)</pre>	<pre>document.writeln(Math.abs(-4)); document.writeln(Math.abs(-7.8)); document.writeln(Math.abs('-4')); document.writeln(Math.abs(null)); //0 document.writeln(Math.abs("string")); //NaN</pre>
<code>sign()</code>	<p>It returns the sign of the given number.</p> <pre>Math.sign(num)</pre>	<pre>document.writeln(Math.sign(12)); //1 document.writeln(Math.sign(-12)); //-1 document.writeln(Math.sign(0)); //0</pre>
<code>sqrt()</code>	<p>It returns the square root of the given number.</p> <pre>Math.sqrt(num)</pre>	<pre>document.writeln(Math.sqrt(16)); document.writeln(Math.sqrt(12)); document.writeln(Math.sqrt(-9)); //NaN</pre>
<code>cbrt()</code>	<p>It returns the cube root of the given number.</p> <pre>Math.cbrt(num)</pre>	<pre>document.writeln(Math.cbrt(8)); document.writeln(Math.cbrt(-64));</pre>
<code>ceil()</code>	<p>It returns a smallest integer value, greater than or equal to the given number.</p> <pre>Math.ceil(num)</pre>	<pre>document.writeln(Math.ceil(7.2)); //8 document.writeln(Math.ceil(0.2)); //1 document.writeln(Math.ceil(-7.2)); // -7 document.writeln(Math.ceil(-0.2)); //0</pre>

<u>floor()</u>	<p>It returns largest integer value, lower than or equal to the given number.</p> <p>Math.floor(num)</p>	<pre>document.writeln(Math.floor(7.2)); //7 document.writeln(Math.floor(0.2)); //0 document.writeln(Math.floor(-7.2));//-8 document.writeln(Math.floor(-0.2)); //-1</pre>
<u>round()</u>	<p>It returns closest integer value of the given number.</p> <p>Math.round(num)</p>	<pre>document.writeln(Math.round(7.2)); //7 document.writeln(Math.round(0.6)); //1 document.writeln(Math.round(-7.2)); // -7 document.writeln(Math.round(-0.6)); // -1</pre>
<u>trunc()</u>	<p>It returns an integer part of the given number.</p> <p>Math.trunc(num)</p>	<pre>document.writeln(Math.trunc(12.24)); //12 document.writeln(Math.trunc(0.84)); //0 document.writeln(Math.trunc(-12.24)); // -12 document.writeln(Math.trunc(-0.84)); //0</pre>
<u>max()</u>	<p>It returns maximum value of the given numbers.</p> <p>Math.max(num1,num2,...,numN)</p>	<pre>document.writeln(Math.max(22,34,12)); //34 document.writeln(Math.max(-10,-24,-12)); // -10</pre>
<u>min()</u>	<p>It returns minimum value of the given numbers.</p> <p>Math.min(num1,num2,...,numN)</p>	<pre>document.writeln(Math.min(22,34,12)); //12 document.writeln(Math.min(-10,-24,-12)); // -24</pre>
<u>pow()</u>	<p>It returns value of base to the power of exponent.</p> <p>Math.pow(base,exponent)</p>	<pre>document.writeln(Math.pow(2,3)); document.writeln(Math.pow(5,2.4)); document.writeln(Math.pow(2,-3));</pre>
<u>random()</u>	<p>It returns random number between 0 (inclusive) and 1 (exclusive).</p> <p>Math.random()</p>	<pre>document.writeln(Math.random())</pre>
<u>sin()</u>	<p>It returns the sine of the given number.</p> <p>Math.sin(num)</p>	<pre>document.writeln(Math.sin(1));</pre>

<u>cos()</u>	<p>It returns the cosine of the given number.</p> <p>Math.cos(num)</p>	document.writeln(Math.cos(1));
<u>tan()</u>	<p>It returns the tangent of the given number.</p> <p>Math.tan(num)</p>	document.writeln(Math.tan(1));
<u>sinh()</u>	<p>It returns the hyperbolic sine of the given number.</p> <p>Math.sinh(num)</p>	document.writeln(Math.sinh(-1)); document.writeln(Math.sinh(0));
<u>cosh()</u>	<p>It returns the hyperbolic cosine of the given number.</p> <p>Math.cosh(num)</p>	document.writeln(Math.cosh(-1)); document.writeln(Math.cosh(0));
<u>tanh()</u>	<p>It returns the hyperbolic tangent of the given number.</p> <p>Math.tanh(num)</p>	document.writeln(Math.tanh(-1))

The Math object constants /Properties

Property Name	Description	Example
LN2	Natural logarithm of 2, approximately 0.693.	var x= Math.LN2
LN10	Natural logarithm of 10, approximately 2.302.	var x=Math.LN10
LOG2E	Base 2 logarithm of E, approximately 1.442.	var x=Math.LOG2E
LOG10E	Base 10 logarithm of E, approximately 0.434.	var x=Math.LOG10E
PI	Ratio of the circumference of a circle to its diameter, approximately 3.14159.	var x=Math.PI
SQRT1_2	Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.	var x=Math.SQRT1_2
SQRT2	Square root of 2, approximately 1.414.	var x=Math.SQRT2

Number Object

The JavaScript number object **enables you to represent a numeric value**. It may be integer or floating-point. JavaScript number object follows IEEE standard to represent the floating-point numbers.

The constructors for Number Object is

`Number()`

`Number(value)`

Eg: var `n=new Number(16);`

The Number object constants /Properties

MIN_VALUE	returns the largest minimum value.
MAX_VALUE	returns the largest maximum value.
POSITIVE_INFINITY	returns positive infinity, overflow value.
NEGATIVE_INFINITY	returns negative infinity, overflow value.
NaN	represents "Not a Number" value.

The Number object methods are

Method Name	Description	Example
<u>isFinite()</u>	It determines whether the given value is a finite number. <code>Number.isFinite(num)</code>	<code>var x=0; var y=11; var z=-111; document.writeln(Number.isFinite(x)); //true document.writeln(Number.isFinite(y)); //true document.writeln(Number.isFinite(z)); //true</code>
<u>isInteger()</u>	It determines whether the given value is an integer. <code>Number.isInteger(num)</code>	<code>var y=1; document.writeln(Number.isInteger(y)); //true</code>

<u>parseInt()</u>	<p>It converts the given string into an integer number.</p> <pre>Number.parseInt(string, radix)</pre> <pre>document.writeln(Number.parseInt(a,10)); //decimal value</pre> <pre>document.writeln(Number.parseInt(a,8)); //octal value</pre> <pre>document.writeln(Number.parseInt(a,16)); //hexa decimal value</pre>	<pre>var a="50"; var d="50String"; var b="welcome2019";</pre> <pre>document.writeln(Number.parseInt(a)); //50 document.writeln(Number.parseInt(d)); //50 document.writeln(Number.parseInt(b)); //NaN</pre>
<u>parseFloat()</u>	<p>It converts the given string into a floating point number.</p> <pre>Number.parseFloat(string)</pre>	<pre>var b="50.25" var d="50.25String" document.writeln(Number.parseFloat(b)); document.writeln(Number.parseFloat(d));</pre>
<u>toString()</u>	<p>It returns the given number in the form of string.</p> <pre>toString()</pre>	<pre>var n=new Number(16); document.writeln(n.toString());</pre>
<u>valueOf()</u>	valueOf()	<pre>document.writeln(n.valueOf());</pre>
<u>toExponential()</u>	<p>It returns the string that represents exponential notation of the given number.</p> <pre>Number.toExponential(num)</pre>	<pre>var a=989721; document.writeln(a.toExponential()); 9.89721e+5</pre>

Boolean Object

The **Boolean** object represents two values, either "true" or "false".

If **value** parameter is omitted or is 0, -0, null, false, NaN, undefined, or the empty string (""), the object has an initial value of false.

```
var val= new Boolean(value);
```

```
var x = Boolean(10>9) ; // true
```

```
var val= new Boolean(true); //true
```

```
var val= new Boolean(0); //false
```

```

var x= false;
var y= new Boolean(false); // (x == y) is true because x and y have equal
values

var x = false;
var y = new Boolean(false);

// (x === y) is false because x and y have different types

```

The Boolean object methods are

Method Name	Description	Example
toString()	This method returns a string of either "true" or "false" depending upon the value of the object. boolean.toString()	var flag = new Boolean(false); document.write("flag.toString is : " + flag.toString());
valueOf()	Returns the primitive value of the specified Boolean object. boolean.valueOf()	. var flag = new Boolean(false); document.write("flag valueOf is : " + flag.valueOf());

DOM (Document Object Model)Objects

1. window Object

- window represents the web browser open window.
- If webpage is divided into frames, each frame corresponds to a separate window object.

The properties of window object are

Property Name	Description
closed	Returns a Boolean value indicating whether a window has been closed or not
defaultStatus	Sets or returns the default text in the status bar of a window
document	Returns the Document object for the window
history	Returns the History object for the window
location	Returns the Location object for the window
name	Sets or returns the name of a window
navigator	Returns the Navigator object for the window
outerHeight	Returns the height of the browser window, including toolbars/scrollbars
outerWidth	Returns the width of the browser window, including toolbars/scrollbars
status	Sets or returns the text in the status bar of a window

The methods of window Object are

Method Name	Description	Example
alert()	Displays an alert box with a message and an OK button. alert(message)	<code>alert("Hello Welcome to Javascript");</code>
confirm()	Displays a dialog box with a message and an OK and a Cancel button. A confirm box is often used if you want the user to verify or accept something. confirm(message)	<code>confirm("Perform or not?");</code>
prompt()	Displays a dialog box that prompts the visitor for input. prompt(text, defaultText)	<code>prompt("enter a no", "12")</code>
blur()	Removes focus from the current window. window.blur()	<code>myWindow.blur();</code>

focus()	Sets focus to the current window. window.focus()	myWindow.focus();
close()	Closes the current window. window.close()	myWindow.close();
open()	<p>Opens a new browser window. window.open(<i>URL, name, specs, replace</i>);</p> <p>URL- Optional, any URL. name – Optional , the value can be _blank,_top,_parent,_self, name of window.</p> <p>Specs - can be</p> <ul style="list-style-type: none"> fullscreen=yes no 1 0, height=pixels, status=yes no 1 0, scrollbars= yes no 1 0, resizable= yes no 1 0, titlebar=yes no 1 0, toolbar=yes no 1 0, top=pixels, width=pixels. <p>Replace – can be</p> <ul style="list-style-type: none"> true - URL replaces the current document in the history list. false - URL creates a new entry in the history list 	<pre>window.open("https://www.google.com");</pre> <pre>var myWindow = window.open("", "MsgWindow", "width=200,height=100");</pre> <pre>var myWindow = window.open("", "_self");</pre> <pre>window.open("https://www.google.com", "_blank", "toolbar=yes,scrollbars=yes,resizable=yes,top=500,left=500,width=400,height=400");</pre>
moveTo()	Moves a window to the specified position window.moveTo(<i>x, y</i>)	myWindow.moveTo(500, 100);
print()	Prints the content of the current window. window.print(). //a default printer is assigned to get current page print.	window.print()

2. document Object

- document object represents the HTML document or webpage that is currently opened in the web browser.
- When an HTML document is loaded into a web browser, it becomes a document object.

The properties of document object are

Property Name	Description
activeElement	Returns the currently focused element in the document
anchors	Returns a collection of all <a> elements in the document that have a name attribute
applets	Returns a collection of all <applet> elements in the document
body	Sets or returns the document's body (the <body> element)
close	Closes the output stream previously opened with document.open()
cookie	Returns all name/value pairs of cookies in the document
doctype	Returns the Document Type Declaration associated with the document
documentElement	Returns the Document Element of the document (the <html> element)
title	Sets or returns the title of the document
URL	Returns the full URL of the HTML document
lastModified	Returns the date and time the document was last modified

The methods of document Object are

Method Name	Description	Example
getElementById()	Returns the element that has the ID attribute with the specified value. document.getElementById(elementID) elementID : The ID attribute's value of the element you want to get	<code>document.getElementById("101");</code>
getElementsByName()	Returns a NodeList containing all elements with a specified name. document.getElementsByName(name)	<code>var x = document.getElementsByName("fname");</code>
getElementsByTagName()	Returns a NodeList containing all elements with the specified tag name. document.getElementsByTagName(tagname)	<code>var x = document.getElementsByTagName("LI");</code>

getElementsByClassName()	Returns a NodeList containing all elements with the specified class name. document.getElementsByClassName(classname)	var x = document.getElementsByClassName("example");
write()	Writes HTML expressions or JavaScript code to a document. document.write(exp1, exp2, exp3, ...)	document.write("Hello World!");
writeln()	Same as write(), but adds a newline character after each statement. document.writeln(exp1, exp2, exp3, ...)	document.writeln("Hello World!");

3. history Object

- Represents a list of URLs that the user has already accessed.
- If the webpage uses frames, each frame separately has its own history object .i.e. each has its own list of URLs.

The properties of history object are

Property Name	Description
length	Returns the number of URLs in the history list

The methods of history Object are

Method Name	Description	Example
back()	Loads the previous URL in the history list. history.back()	function goBack() { window.history.back(); }
forward()	Loads the next URL in the history list. history.forward()	function goForward() { window.history.forward(); }
go()	Loads a specific URL from the history list. history.go(number URL) back -> -ve forward-> _ve	function goBack() { window.history.go(-2); }

4. navigator Object

- Allows to access various information about user's web browser.

The properties of navigator object are

Property Name	Description
appCodeName	Returns the code name of the browser.
appName	Returns the name of the browser
appVersion	Returns the version information of the browser
cookieEnabled	Determines whether cookies are enabled in the browser
platform	Returns for which platform the browser is compiled
language	Returns the language of the browser
product	Returns the engine name of the browser

The methods of navigator Object are

Method Name	Description	Example
javaEnabled()	Specifies whether or not the browser has Java enabled. <code>navigator.javaEnabled()</code>	<code>var x = "Java Enabled: " + navigator.javaEnabled();</code>

5. location Object

- Allows to access information about the URL of the webpage that is currently opened in a window or frame.

The properties of location object are

Property Name	Description
host	Sets or returns the hostname and port number of a URL
hostname	Sets or returns the hostname of a URL
href	Sets or returns the entire URL
pathname	Sets or returns the path name of a URL
port	Sets or returns the port number of a URL
protocol	Sets or returns the protocol of a URL
search	Sets or returns the "query string" part of a URL

The methods of location Object are

Method Name	Description	Example
assign()	Loads a new document. <code>location.assign(<i>URL</i>)</code>	<code>location.assign("https://www.gmail.com");</code>
reload()	Reloads the current document <code>location.reload(<i>flag</i>)</code> flag can be <ul style="list-style-type: none"> • false - Default. Reloads the current page from the cache. • true - Reloads the current page from the server 	<code>location.reload();</code>
replace()	Replaces the current document with a new one. <code>location.replace(<i>newURL</i>)</code>	<code>location.replace("https://www.google.com");</code>

6. screen Object

- Allows to access different information about the user's screen

The properties of screen object are

Property Name	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)
colorDepth	Returns the bit depth of the color palette for displaying images
height	Returns the total height of the screen
width	Returns the total width of the screen
pixelDepth	Returns the color resolution (in bits per pixel) of the screen

JavaScript Regular Expressions

A regular expression is an object that describes a pattern of characters.

The JavaScript **RegExp** class represents regular expressions, and both **String** and **RegExp** define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

RegExp object

Regular expressions are used to perform pattern-matching and "search-and-replace" functions on text.

The syntax for defining a pattern is

/pattern/modifiers;

Modifiers

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

S.No	Modifier	Description	Example
1	<i>g</i>	Perform a global match (find all matches rather than stopping after the first match)	var ptr = /cse/g;
2	<i>i</i>	Perform case-insensitive matching	var ptr = /cse/i;
3	<i>m</i>	Perform multiline matching	

Brackets

Expression	Description	Example
[abc]	Find any character between the brackets	<pre>var str = "Welcome2019 July 18"; var patr = /[mn]/gi; var result = str.match(patr); document.writeln(result);</pre>
[^abc]	Find any character NOT between the brackets	<pre>var str = "Welcome2019 July 18"; var patr = /[^mnJW9]/gi; var result = str.match(patr); document.writeln(result);</pre>
[0-9]	Find any character between the brackets (any digit)	<pre>var str = "Welcome2019 July 18"; var patr = /[7-9]/g; var result = str.match(patr); document.writeln(result);</pre>
[^0-9]	Find any character NOT between the brackets (any non-digit)	<pre>var str = "Welcome2019 July 18"; var patr = /[^7-9]/g; var result = str.match(patr);</pre>
(x y)	Find any of the alternatives specified	<pre>var str = "Welcome2019 July 18"; var patr= /(c e)/igm; var result = str.match(patr); document.writeln(result);</pre>
[A-Z]	Any character from uppercase A to uppercase Z	<pre>var str = "Welcome2019 July 18"; var patr = /[I-X]/g; var result = str.match(patr); document.writeln(result);</pre>
[a-z]	Any character from lowercase a to lowercase z	<pre>var str = "Welcome2019 July 18"; var patr = /[c-v]/g; var result = str.match(patr);</pre>
[A-z]	Any character from uppercase A to lowercase z	<pre>var str = "Welcome2019 July 18"; var patr = /[P-m]/g; var result = str.match(patr); document.writeln(result);</pre>

Quantifiers

Expression	Description	Example
n+	Matches any string that contains at least one n	<code>var str = "Helloo Welcome!"; var patt1 = /o+/g; var result = str.match(patt1); document.writeln(result);</code>
n*	Matches any string that contains zero or more occurrences of n	<code>var str = "1, 100 or 1000?"; var patt1 = /10*/g; var res=str.match(patt1); document.writeln(res);</code>
n?	Matches any string that contains zero or one occurrences of n	<code>var str = "1, 100 or 1000?"; var patt1 =new RegExp("10?", "g"); var res=str.match(patt1); document.writeln(res);</code>
n{X}	Matches any string that contains a sequence of X n's	<code>var str = "1, 10, 100 or 1000?"; var patt1 =new RegExp("0{2}", "g"); var res=str.match(patt1); document.writeln(res);</code>
n{X,Y}	Matches any string that contains a sequence of X to Y n's	<code>var str = "1, 10, 100 or 1000?"; var patt1 =new RegExp("0{1,3}", "g"); var res=str.match(patt1); document.writeln(res);</code>
n\$	Matches any string with n at the end of it	<code>var str = "Is cse\nthis cse\nhis cse\n"; var ptr = /cse\$/gm; var res = str.match(ptr); document.writeln(res);</code>
^n	Matches any string with n at the beginning of it	<code>var str = "cs Is \ncs this \ncs his\n"; var ptr = /^cs/gm; var res = str.match(ptr); document.writeln(res);</code>

Metacharacters

Expression	Description	Example
•	Find a single character, except newline or line terminator	<code>var str = "CSE ece eee!"; var patt1 = /e.e/g; var result = str.match(patt1);</code>
\w	Find a word character. i.e. only alphanumeric character.	<code>var str = "@welcome 2009#!"; var patt1 = /\w/g; var result = str.match(patt1); document.writeln(result);</code>
\W	Find a non-word character. i.e. Non alphanumeric Character.	<code>var str = "@welcome 2009#!"; var patt1 = /\W/g; var result = str.match(patt1); document.writeln(result);</code>
\d	Find a digit	<code>var str = "@welcome 2009#!"; var patt1 = /\d/; var result = str.match(patt1); document.writeln(result); //2</code>
\D	Find a non-digit	<code>var str = "@welcome 2009#!"; var patt1 = /\D/; var result = str.match(patt1); document.writeln(result); //@</code>
\S	Find a non-whitespace character	<code>var str = "@welcome 2009#!"; var patt1 = /\S/g; var result = str.match(patt1);</code>
\s	Find a whitespace character	<code>var str = "@welcome 2009 #!"; var patt1 = /\s/g; var result = str.match(patt1); document.writeln(result + ",");</code>
\n	Find a new line character	<code>var str = "wel\ncome"; var patt1 = /\n/; var result = str.search(patt1); document.writeln(result + ",");</code>

RegExp Object Methods

Method	Description	Example
compile()	Compiles a regular expression	<pre>var str="Every man and Every woman on earth are Great"; patt=/wo?man/g; patt.compile(patt); str=str.replace(patt, "person"); document.write(str); // Every person and Every person on earth are Great</pre>
exec()	Tests for a match in a string. Returns the first match	<pre>var str = "The World is beautiful"; var patt = new RegExp("e"); var res = patt.exec(str); document.writeln(res); //e</pre>
test()	Tests for a match in a string. Returns true or false	<pre>var str = "Hello world!"; var patt = /Hello/g; var res = patt.test(str); document.writeln(res); //true patt2 = /wel/g; res = patt2.test(str); document.writeln(res); //false</pre>
toString()	Returns the string value of the regular expression	<pre>var patt = new RegExp("Hello World", "g"); var res = patt.toString(); document.writeln(res);</pre>
match(regexp)	The match() method searches a string for a match against a regular expression, and returns the matches, as an Array object.	<pre>var str= "The reason for season in common places"; var res = str.match(/on/gi); document.writeln(res);</pre>

Dynamic HTML with Javascript

Date:
Page:

- Javascript events provide interactive webpages.

- Dynamic HTML makes webpages come alive and allows to respond to user's actions.

- JavaScript uses event handling / events for DHTML.

Event - An event is a state change

Eg: Mouse click, keypress etc.

An event can occur with user intervention (or) without .

Eg: mouseclick, on page load etc.

- Javascript defines several in built events to assist in making webpages dynamic & interactive.

"When an event occurs, its appropriate event handler is executed/called."

"An event handler is a special function that handles a particular event."

Java script allows to create an event handler and associate it with the desired event.
The event handlers are associated with events through certain attributes of various HTML tag.

The Common events & event handling attributes

<u>Event</u>	<u>Description</u>	<u>Attribute</u>
abort	Occurs when user stops or cancels the loading of an image.	[onabort]
blur	Occurs when an HTML element loses focus bcz of user clicks outside of the element	[onblur]
change	Occurs when user change the value of a textfield	[onchange]
click	Occurs when user click on HTML element like form element, image, link, buttons etc.	[onclick]
dblclick	Occurs when user double click on a HTML element	[ondblclick]
error	Occurs when an error occurs while page or image is loaded	[onerror]
focus	Occurs when an HTML element gets input focus	[onfocus]
keypress	Occurs when user pressed a key	[onkeypress]
keyup	Occurs when user released a key	[onkeyup]
keydown	Occurs when user has pressed a key	[onkeydown]
load	Occurs when a page or image is completely loaded in the Web browser	[onload]

Event

Description

Attribute

mousedown

Occurs when the user has just pressed a mouse button

onmousedown

mouseover

Occurs when the user moves the mouse on top of a form element

onmouseover

mouseup

Occurs when the user has just released a mouse button

onmouseup

select

Occurs when the user selects a text / text area field in HTML form

onselect

submit

Occurs when the user submits an HTML form by pressing submit button

onsubmit

mousemove

Occurs when mouse moves

onmousemove

mouseout

Occurs when the mouse leaves an element

onmouseout

onreset

Occurs when user clicks the reset button

onreset

onresize

Occurs when an element or page is resized.

onresize

onunload

Occurs when a page is unloaded

onunload

Introduction to PHP

Date _____
Page _____

PHP - (Hypertext Preprocessor) is a Server side Scripting language used to Create dynamic webpages.

- Originally called Personal Home page
- The various versions of PHP are PHP version 2 — PHP version 6. (Current)
- PHP Scripts are executed on Server.
- PHP is an open source software
- Developed by Apache Group, in 1994.

PHP Characteristics

- * Server side Scripting language
- * Open source Software
- * Free to download and use. (www.php.net)
- * Runs on different platforms. (Windows, Linux etc)
- * Compatible with all the servers used today
(Apache, IIS etc)
- * Supports many databases
(MySQL, Oracle, Sybase, PostgreSQL etc)
- * A PHP file contains text, HTML tags and script
- * PHP files are returned to the browser as HTML.
- * PHP files have extension ".php", ".php3" or ".phtml".
- * Mainly used for form handling & database access.
- * Processes script code in Interpreter & PHP Parser.
- * PHP statements are terminated by Semicolon (;

Features of PHP.

- PHP is a general purpose scripting language runs on a webserver.
- PHP takes script code as input and create webpage content dynamically, as output.
- PHP can be deployed on most of the webservers and on almost every operating system, platform.

The Main features of PHP are

1. Access control - provides a built in webbased configuration screen to handle access control configuration.
2. File upload support - PHP provides MIME (multipurpose Internet Mail Extensions) decoding process to upload files on to the server. Allows users to upload files to a webserver.
3. HTTP - based Authentication control - allows users can create customized authentication mechanisms for web server.
4. Supports Variables, arrays , Associative arrays. can be passed from one webpage to another using GET & POST method forms.
5. Conditional statements and loops
6. Extended regular expressions - used for pattern matching, Pattern substitution & string manipulation.

7. Raw HTTP Header Control - Allows to transfer a URL from one client to another client.
Used to manipulate latest updated header of webpage.
8. Access Logging - Allows to record no. of times a webpage or website is accessed. It also helps to generate footer on every page, displaying access information.
9. Safe mode Support - Allows multiple users to run PHP scripts on the same server simultaneously. Solves Shared-server Security problem.
10. Open Source - Allows the user to work with different software development languages. User can choose a language to create its own source code for different types of applications and distribute them on www free of cost.
11. Third Party application support - Supports a wide range of different databases, include a common API for database access.
12. PHP's Extensible Architecture. - Allows the user to read and write in various formats, such as GIF, JPEG, PNG, Send & receive mails using SMTP, Internet Protocols. Allows to access C libraries, Java classes etc.

13. Dynamic Typing - No need to declare variables

in PHP. The type of variable gets set only when it is assigned with some value.

14. Large no. of library functions - to develop effective code in PHP.

Advantages of PHP over Other Scripting languages.

1. Active Server pages (ASP) -

- ASP } - Supported by only Microsoft IIS. (Internet Information Server (IIS), used only for Win32 (Windows) system.
- ASP } - ASP is much slower, less stable & less secure.

- PHP } - supported by almost all the web servers.
- PHP } - works with apache, proven record of speed, reliability & secure.

2. Cold Fusion - Available only for Win32, Solaris, Linux & HP, designed for the non programmers.

- PHP } - focused on programmers.
- PHP } - Compared with cold fusion, faster, more efficient & stable language.

3. Perl (Practical Extraction & Reporting language)

- Relatively PHP is easier to integrate with HTML
- Easy to learn & has smaller learning curve.
- No prior knowledge of programming required.
- Perl is for complex tasks, more complicated.

4. JSP (Java server Pages)

- PHP is supported for all platforms, that is equal or above 32 bits.
- But JSP is supported by only those platforms that have a JVM.
- Performance of PHP is 5 times faster than JSP.

Creating a PHP Script

```
demo.php {  
    <html>  
        <body>  
            <?php  
                echo "Introduction to PHP"  
                echo "Welcome to server side scripting"  
            ?>  
        </body>  
    </html>  
}
```

The PHP file contains PHP instructions, Javascript, HTML & CSS.

On Browser → http://localhost/demo.php

- The Webserver receives the request, gives the PHP Script to PHP Parser and Interpreter for further processing.
- The Interpreter reads script in `<?php ?>`, executes them, passes the result back to Server, the Server sends the result back to the browser.
- 'echo' or 'print' statement used to display.

1. PHP code must be enclosed within
`<?php ... ?>` Tags
2. Every PHP statement must end in Semicolon
3. Blank lines in script are ignored by Parser
4. `//` → Single line Comments
`/* ... */` → Multiline Comments.
5. The PHP parser identifies if any error.

Escape Characters

Used to display certain characters.

- `\"` → prints next character as double quote
- `\'` → prints next character as single quote
- `\n` → prints a new line character
- `\t` → prints a tab character
- `\$` → prints a \$ as next character
- `\\"` → prints \ as a next character
- `\r` → prints carriage return as next char.

PHP Variables

Date
Page

Variables and Constants are used to store data to process programs.

A variable has an identifier, value, Scope and lifetime.

Naming Rules & Conventions

① Every variable must be preceded by '\$' sign.

Eg: \$sum, \$name

② Variable must start with a letter or underscore. Eg: \$_sum, \$name, \$123 X

③ Variable name must contain alpha-numeric characters and underscores. No special characters allowed.

Eg: \$sum_60 ✓

\$ sum#60 X, \$@name X

④ The variable name should not contain spaces. More than one word, separate with an underscore | Capitalization.

Eg: \$cse avg X

\$ cseAvg ✓

\$ cse-avg ✓

⑤ No limit on the length of variable name.

⑥ Variables in PHP are case sensitive.

Assigning values to variables.

`$sum = 20;`

`$s = $sum;`

`$s = $s + 1;`

Eg: `$x = 10;`

`$y = 20;`

`echo $x, $y;` → echo statement allows multiple arguments

separated by comma(,)

→ echo statement does not return a value.

✗ `Print $x, $y;` → Not allowed.

Print statement does not

✓ `print $x;`

take multiple arguments

→ Formatted op is possible only with print statement

Assigning a value by Reference method.

<?php

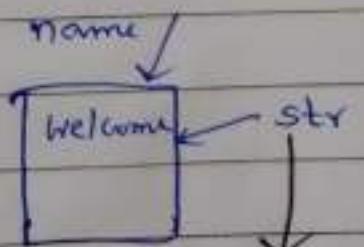
`$name = "Welcome";`

`$str = &$name;`

`$str = "Goodbye";`

~~`$echo $str; //Goodbye`~~

~~`$echo $name; //Goodbye`~~



Reference val

NOTE: Do not assign expression to Reference variables

PHP allows to set the name of a Variable dynamically . i.e.

```
$name = 'student-name'; // Creates new variable  
${$name} = 'Ramesh';  
echo "$student-name"; // Ramesh
```

Destroying Variables

PHP allows to destroy Variable or an element from an array when not required .

The `unset()` function is used to destroy the elements of array or variable .

```
Eg: $name = "Bangalore";  
echo "$name"  
unset($name); // destroy var  
echo '$name'; // error.
```

Using Constants

Constants are identifiers that store values, which can not change during the execution of script.

- Constants are used for defining

1) Configuration settings

2) Error codes and flags

- Constants are case sensitive.

- Constants identifier name is always in uppercase.

- Unlike variable, Constant names do not start with '\$' sign.

- The naming convention is same as variable

- A constant name starts with a letter or underscore, followed by any no. of digits, letters (or) underscore

The syntax to create a constant is

define ("CONSTANT_NAME", Constant_value);

Eg: define ("UNAME", scott);

A constant can hold only a scalar value, whereas a variable can store arrays or objects.

PHP defines several predefined constants.

Eg: M_PI → Mathematical PI

PHP_VERSION → Current Version of PHP.

To access the constant

echo UNAME;

PHP Reserved Keywords

and break case class continue
default do else elseif extends
false for foreach function global
if include list new not
or require return static switch
this true var virtual while
XOR

Exploring Data Types in PHP.

A data type define the type of variable's data.
i.e. category of values.

Integers - Represents a whole number with no fractional components.

The range of integers varies from -95 to 95.

General range -2147483648 to
+2147483647.

Integers also written in decimal,
octal or hexadecimal.

Floating Point Numbers - Represents ~~decimal~~ real numbers that include decimal place.

- 1) numeric floating point no (3.142)
- 2) scientific notation (0.214E2)

The notation is [num]E [exponent].

Strings - • Represents text literals of any length.
• String are enclosed in single quotes
or double quotes.

\$x/y
\$echo

Eg:

\$name = "Hyderabad";

name is Hyderabad ← echo "Name is \$name"
name is \$name ← echo 'Name is \$name'

- Booleans - Represents a true or false value.
- All conditions return true/false based on the conditions tested.
 - Returns always false for
 - ① Keyword false
 - ② integer 0
 - ③ float 0.0
 - ④ Empty string ("")
 - ⑤ String "0"
 - ⑥ An object with no values/methods
 - ⑦ The null value .

- Array - Represents a variable that holds a collection of related data elements.
- / Each element is accessed with index.
 - / Position is specified numerically or alphabetically .

- Object - Stores data and information to process the data . Objects are created by declaring a class .

- Resource - Stores references to functions and resources external to PHP.
- Eg: a database call.

- NULL - , It can have only one value, null.
- / Null is datatype and also a keyword literal . When no value assigned to a variable , it is assigned to null .

* Variable in PHP are not necessary to specify the type, the type is assigned when a value is stored.

Eg: \$name = "Hyderabad"; // String

Datatypes are divided as

① Scalar (or) Primitive datatypes

Integers, floating point nos, Strings and Booleans

② Compound datatypes

Arrays, Objects

③ Special datatypes.

Resources, Null.

Determining Datatype of a variable

gettype() - function is used to return the datatype of a variable dynamically.

Eg: \$value = "Hyderabad";

echo gettype(\$value); // String

unset(\$value); \$value = "31.72";

gettype(\$value); echo gettype(\$value); // double

echo gettype(0); // null

\$value = 3

echo gettype(\$value); // integer

→ null



The Special Functions to check datatype

- is_bool() — Tests if a variable holds a Boolean value
- is_numeric() — Tests if a variable holds a numeric value
- is_int() — Tests if a var holds a integer
- is_float() — Tests if a var holds a float
- is_string() — Tests if a var holds a string
- is_null() — Tests if a var holds a NULL
- is_array() — Tests if a var is an array
- is_object() — Tests if a var is an object

- PHP is an open source , interpreted , Object oriented Scripting language .
- Used to develop dynamic webpages
- Executes Script code on the Server.
- PHP Script executes much faster than other Scripts
- Platform independant
- Compatible will all servers Apache, IIS etc
- PHP code is easily embedded with HTML Tags & script.
- PHP is a loosely typed language , i.e. PHP automatically converts the variable to its correct data type.

13. Dynamic Typing - No need to declare variable in PHP. The type of variable gets set only when it is assigned with some value.
14. Large no. of library functions - to develop effective code in PHP.

Advantages of PHP over Other Scripting languages.

1. Active Server pages (ASP) -

- } - Supported by only Microsoft IIS. (Internet Information Server (IIS), used only for Win32 (Windows) system.
- ASP is much slower, less stable & less secure.

- } - supported by almost all the web servers.
- works with apache, proven record of speed, reliability & secure.

2. Cold fusion - Available only for Win32, Solaris, Linux & HP, designed for the non programmers.

- } - focused on programmers.
- Compared with cold fusion, faster, more efficient & stable than language.

3. Perl (Practical Extraction & Reporting language)

- Relatively PHP is easier to integrate with HTML, easy to learn & has smaller learning curve.
- No prior knowledge of programming required.
- Perl is for complex tasks, more complicated.

4. JSP (Java server Pages)

- PHP is supported for all platforms, that is equal or above 32 bits.
- But JSP is supported by only those platforms that have a JVM.
- Performance of PHP is 5 times faster than JSP.

Operators in PHP

Date _____
Page _____

1. Assignment operators ($=$)

= (equal) operator is used to assign value to variable (or) to assign one variable to another.

Eg: $\$avg = 31.42;$

$\$x = \$y;$

$\$sum = \$x + \$y;$

2. Arithmetic operators

Used to perform basic mathematical operations.

| | |
|--------------|------------------|
| $-\$x$ | - Negation |
| $\$x + \y | - Addition |
| $\$x - \y | - Subtraction |
| $\$x * \y | - Multiplication |
| $\$x / \y | - Division |
| $\$x \% \y | - Modulus |

} Arithmetic operators

| | | |
|---------------------------------|----------|--|
| Arithmetic Assignment operators | $+=$ | - Adds value & assigns to a variable |
| | $-=$ | - subtracts the value & " " " |
| | $*=$ | - multiplies the value & " " * |
| | $/=$ | - divides the value & " " * |
| | $\%=$ | - divides & assigns modulus " " % |
| | $\cdot=$ | - concatenates and assigns the value to a variable. (only for strings) |

Eg: $\$x = 9;$ $\$x += 3;$

$\$x = "hello";$ $\$x .= "hi";$ // hello hi.

3. String Operators (.)

- The dot (.) string concatenation operator is used to combine values to create a string.
- Build a string from other strings, variables and numbers.

Eg: ① \$str = "Hyderabad";

echo 'my city is' . \$str;

② \$ echo 'my city is' . 'hyd';

4. Comparison Operators

Used to compare one value with another and returns true or false depending on status of the match.

== - Equal to

!= - Not equal to

<> - Not equal to

==> - Identical to → { True if first opnd equal to second operand

! ==> - Not Identical to in both value & type

< - Less than

> - Greater than

<= - Less than or equal to

>= - greater than or equal to

5. Logical operators.

Also called boolean operators, evaluates the logical expression and return true or false.

| | | |
|-----|---|-----------------------|
| and | - | Logical AND operation |
| or | - | Logical OR operation |
| xor | - | Logical XOR operation |
| && | - | Logical AND operation |
| | - | Logical OR operation |

6. Increment | Decrement Operators

| | | |
|-------|---|----------------|
| \$x++ | - | Post increment |
| ++\$x | - | Pre increment |
| \$x-- | - | Post decrement |
| --\$x | - | Pre decrement |

Eg:

$\$x = 10 ;$

$\$y = \$x++ ;$

echo \$y ; // 10

$\$y = ++\$x ;$

echo \$y ; // 12

$\$y = \$x-- ; // 12$

$\$y = --\$x ; // 10$

Operator Precedence

Associativity

Right

— !

logical

Left

— *, /, %

Arithmetic

Left

— +, -, •

Arithmetic & String

Left

— &

Bitwise, Reference

Left

— |

Bitwise

Left

— &&

Logical

Left

— ||

Logical

Right

— =+, =-, =*

Assignment

=!, =., =&,

=!, =^, =<<,

=>>.

Left

— and

logical

Left

— xor

logical

Left

— or

logical

Left

— ,

7. Bitwise operators

Performs operations at bit level.

& — Bitwise AND

| — Bitwise OR

^ — Bitwise XOR

! — Bitwise NOT.

Type Casting

Implicit Conversion

PHP allows to convert datatype of a variable to another datatype, is called "type casting".

Eg:

- ① $\$x = 35.42; \$y = 20;$
 $\$z = \$x + \$y; // float.$
- ② $\$x = "4"; // string$
 $\$x = \$x + 3; // int, integer$
 $\$x = \$x + 3.5; // 10.5, float or double$

Explicit type conversion

1. **`Var = (target-type) variable`**

`$\$x = (boolean) \$x // convert to boolean$`

`$\$x = (float) \$x // convert to float$`

Eg:

`$\$x = 34.5;$`

`$\$y = 20;$`

`$\$z = \$x + \$y; // float - 54.5;$`

`$\$z = (int) \$x + \$y; // int - 54.$`

2.

`settype (var-name, "target-type");`

`$\$z = \$x + \$y // float$`

`$settype (\$z, "int"); // int$`

`$echo "\$z"; // 54.$`

Eg:

`$settype (\$x, "array"); // to array$`

`$settype (\$y, "bool"); // to boolean$`

Control Structures

Date _____
Page _____

1. Conditional Statements.

- ① if statement
- ② if - else statement
- ③ if - elseif - else statement
- ④ switch - case statement.

↳ Allows case constant as string also,
including integer constant / char const.

2. Looping (or) Iterative statements

- ① while loop
- ② do-while loop
- ③ for - loop
- ④ foreach loop.

while (condition)
{
statements
}

do {
statements

} while (condition);

for (initialization; condition; inc/dec)
{
statements

(i) foreach (array as value)

{
statements

}

(ii) foreach (array as key \Rightarrow val)

{
statement;

}

3. ~~for~~ Jump statements

(i) break

(ii) Continue

(iii) exit - used when we want to stop
a program from running. It
can block infinite looping statement

Eg:

```
for ($i=0 ; $i < 5 ; $i++)
```

{

```
if ($i == 3)
```

```
{ exit;
```

```
}
```

```
echo "The no is " . $i;
```

```
echo "<br>";
```

```
}
```

```
echo "This is exit";
```

Arrays

Date _____
Page _____

An array is a collection of elements / values of similar type in a single variable.

Array elements are accessed by name and index.

Eg: \$cities = array('hyd', 'kmr', 'nzb', 'vow');
echo "The second name is \$cities[1]";

PHP supports 3 types of arrays.

① Numeric Array - (Indexed Array)

- Defines an array with a numeric ID key (index).
- The index value starts from '0'.
- The array can store number, string, object etc.
- The integer values as their index.

Eg: <?php

\$cities = array("hyd", "chennai", "delhi", "bopal");

print_r(\$cities);

?>

Eg: \$cities[2]

print_r() - It is a built in function in PHP, used to print or display information stored in a variable.

print_r(\$variable)

print_r(\$variable, \$flag)

If flag is true, the result is stored into a variable, rather than printing.

By default flag is false, flag is optional.

Ex: \$cities = array ("hyd", "zoty", "wngi", "dellw");
print_r (\$cities);

\$res = print_r (\$cities, true);
echo "\$res";

② Associative Arrays.

In associative arrays, an index is associated with a value.

Ex: Employee name - key
salary - value.

The values are accessed by specifying key.

Ex. \$sal = array ("Teja" => 35000,
"Ravi" => 60000,
"Sahithi" => 45000);

(or)

\$sal ["Teja"] => 35000;

\$sal ["Ravi"] => 60000;

\$sal ["Sahithi"] => 45000;

3. Multidimensional Arrays.

A multidimensional array is an array of arrays.
i.e. A array can contain sub array, it further
has sub array and so on.

Eg: \$emps = array (

array ("ram" => 87000, "surendra" => 97000,
"ganesh" => 99000),

array ("rajesh" => 165000, "anush" => 115000),

array ("mahesh" => 45000, "hanish" => 35000));

print_r(\$emps);

— — —

```
<!-- using variables and constants in PHP */
```

```
<html>
<body>
<?php
    //accessing variables
    $p = 45;
    $q = 55;
    echo $p , "<br>",$q,"<br>";
    // print $p,$q; // error- not allows two args.
    //variables by reference method
    $x = 20;
    echo "The X value is $x <BR>";
    $y = &$x;
    echo " X is = $x <BR> y is =$y <BR>" ;
    $y = 35;
    echo "y is $y <BR> ";
    echo "x is $x <br> ";
    //dynamicaaly creating variable name
    $name = 'student_name';
    ${$name} = 'ramesh'; //assigning value to dynamic variable
    echo "$student_name is value aasigned to dynamic variable <br>";
    $student_name = 'suresh'; //changing value
    echo "$student_name", "<br> ";
    //unset a variable(destroying).
    $college="Vardhaman";
    echo "The college name is $college <BR>";
    unset($college);
    echo "After unset the value of $college"; //error

    //defining constants in PHP
    echo "<Br>";
    define("UNAME" , "scott");
    define("PI" , 3.142);

    echo UNAME,"<BR>", PI;

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

<-- Implicit type conversions in PHP -->

```
<html>
<body>
<?php
    $x="3";//string
    echo gettype($x);
    echo "->$x";
    echo "<br>";

    $x+=3; //converts string to integer
    echo gettype($x);
    echo "->$x";
    echo "<br>";

    $x=$x+3.2; //converts int to double
    echo gettype($x);
    echo "->$x";
    echo "<br>";
?
</body>
</html>
```

<!-- Explicit Type conversions - Type casting -->

```
<html>
<body>
<?php
    //explicit type casting
    $x = 34.5;
    $y = 12;
    $z = $x + $y;
    echo $z; //float val
    echo "<br>";

    /* $z=(int)$x+$y;
    echo $z; // converts to integer or */
    $z=(int)$x+$y;
    settype($z , "int");
    echo "<br>";
    echo $z;
```

```
echo gettype($z);
echo "<br>";
echo (int)$x; //34
?>
</body>
</html>
```

<!--identical operators in PHP -->

```
<html>
<body>
<?php
    $x=10; //integer
    $y=10.00; //double
    if($x == $y)
        echo "Equal <br> "; //equal

    if($x === $y)
        echo "equal ";
    else
        echo "unqual";           //unqual
    ?>
</body>
</html>
```

<!--Knowing datatype in PHP -->

```
<html>
<body>
<?php

/* gettype() function.
gettype(var_name) */

$name = "Hyderabad";
echo gettype($name);//string
echo "<br>";

$name = 67.89;
echo gettype($name);//double
echo "<br>";
```

```

$name = 67;
echo gettype($name); //integer
echo "<br>";
$name = null;
echo gettype($name); //NULL
echo "<br>";

unset($name);
echo gettype($name); //NULL - error
$x=34.5;
if( is_int($x))
    echo "integer";
else
    echo "float";

// $x = $x . "hi" (.=)

?>
</body>
</html>
<!-- Switch to find grade -->
<?php
    $avg=19.5;
    $ch = $avg/10;
    $ch=(int)$ch;
    echo $ch;
    echo "<br>";
    switch($ch)
    {
        case 10: //the case constant can be int , char or string
        case 9:
        case 8:
        case 7:echo "Distionction";break;
        case 6:echo "first";break;
        case 5:echo "second";break;
        default:echo "fail";
    }

?>

```

1. Userdefined Functions In PHP

- PHP allows to write our own (userdefined) functions for reusability.
- Functions make it easy to use the same piece of code several times in application.
- A function in PHP defined as

```
function <fun-name>(arg1, arg2, ... argn)
{
    // code to execute
    return expn;
}
```

A Function is declared using 'function' keyword.

The Naming Conventions for functions are

- Function names are not case sensitive.
i.e. swap(), Swap(), SWAP() refers same.
- / Function name has only chars, digits & underscore.
- / Function name cannot start with digit
- / Two functions cannot have same name.
Bcz PHP does not support function overloading.
- / Reserved keywords cannot be used as names

Scope of Variables

- The variables used within a function are called local variables, the scope or visibility is local to the function. These can not be viewed or modified outside a function.
- The variables that are declared outside the functions are global variables. Any function can view or modify.
- A Function can modify a global variable by using the keyword "global".

e.g.: function fun1()

{

 global \$x = 15; // global variable

}

Arguments passing Methods

① Pass by value e.g: swap(\$a, \$b).

② Passing arguments by Reference
e.g swap(&\$a, &\$b)

① Recursive Functions

Recursive Function is a function that calls itself repeatedly for a specified condition.

A complex problem is divided into small parts.

Increases the flexibility & adaptability of a function.

During Recursion

- ① Must have an exit condition.
- ② Each recursive call must be different than the one before it.

function fact (\$no)

{

 if (\$no < 2)

 return 1 ;

 else

 return (\$no * fact (\$no - 1));

}

② Variable Functions

PHP supports variable functions.

If a variable name has parentheses appended to it, PHP searches for a function with the same name as variable name and executes it.

Variable functions are used to implement
Callbacks.

Default Parameters / Optional Parameters

When creating functions in PHP it is possible to provide default parameters, so that when a parameter is not passed to the function, it is still available within the function with a pre-defined value. These default values can also be called optional parameters, bcz they do not need to be passed to the function.

- Creating a default parameter in a function is very simple and is like normal variable assignment.
- A function can have any number of default parameters, as its need.
- A array can also be ~~use~~ used as default parameter. The default parameters will be added to the end of the required parameters to avoid conflict. default parameters simplify the programming ~~very~~ necessities.

<?php

Eg: function find (\$a, \$b, \$c=10, \$d=5)
 {

return (\$a + \$b + \$c + \$d);

}

\$x = find (10, 20); // 10+20+10+5

\$y = find (10, 20, 30); // 10+20+30+5

\$z = find (10, 20, 30, 40); // 10+20+30+40.

?>

User Defined Functions

Functions Programs in PHP.

<!-- function example with return value-->

```
<?php

function rever($n)
{
    $sum=0; //local variable
    while($n > 0)
    {
        $r = $n % 10;
        $sum = $sum *10 + $r;
        $n = round($n/10);
    }
    return($sum);
}

$x = 1234; //global variable
$y = rever($x);
echo "The reverese number is " . $y;

?>
```

<!-- Passing arguments by value -->

```
<html>
<body>
<?php
    function swap($a , $b)
    {
        $t = $a;
        $a = $b;
        $b = $t;
        echo "After swapping the values are a=$a ,b= $b <br> "; //20,10
    }
}
```

```
$x=10;  
$y=20;  
echo "Before swapping the values are $x , $y <br>";//10,20  
swap($x , $y);  
echo "After swapping x=$x , y=$y"; //10,20  
  
?>  
</body>  
</html>
```

<!-- Passing arguments by reference-->

```
<html>  
  <body>  
    <?php  
  
      function swap(&$a , &$b) //by reference  
      {  
        $t = $a;  
        $a = $b;  
        $b = $t;  
        echo "After swapping the values are a=$a ,b= $b <br> ";  
      }  
  
    ?>  
  </body>  
</html>
```

```
$x=10;  
$y=20;  
echo "Before swapping the values are $x , $y <br>";  
swap($x , $y);  
echo "After swapping x=$x , y=$y";  
  
?>  
</body>  
</html>
```

<!-- Recursive function -->

```
<html>
  <body>
    <?php
      function fact ($n)
      {
        if($n <2)
          return 1;
        else
          return($n * fact($n-1));
      }

      $x = 5;
      $y = fact($x);
      echo "The factorial of $x is $y";

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

<!-- optional parameters in function arguments -->

```
<html>
  <body>
    <?php
      function add($a , $b , $c=12 , $d=10 )
      {

        return($a+$b+$c);

      }

      $x = add(15,18);
      echo "The result is " . $x;
      echo "<br>";
      $y = add(15,18,20);
      echo "The result is " . $y;
      echo "<br>";
      $z = add(15,18,20,30);
      echo "The result is " . $z;

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

<!-- optional parameters as array in function -->

```
<html>
<body>
<?php
function add( $a = Array(10,20,30,40))
{
    $sum = 0;
    for($i=0; $i < count($a) ; $i++)
        $sum = $sum + $a[$i];
    return $sum;
}

$x = Array(1,2,3,4,5,6,7);
echo "The sum is " .add($x);
echo "<br>";
echo "The sum is " .add(); //uses default array

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

<!-- Variable functions-->

```
<html>
<body>
<?php

function rever($n )
{
    $sum=0;
    while($n != 0)
    {
        $r = $n % 10;
        $sum = $sum *10 + $r;
        $n = (int) $n/10;
    }
    echo "The reverese number is $sum";

}
$funcall ='rever'; //assigning function name to variable
$funcall(234); //calling function with variable name.

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

Built-in Array Functions

- Array functions allow manipulating arrays.
- PHP provides both one dimensional and multi dimensional arrays.
- The built in functions provided for arrays are

Function	Description
count()	Counts all elements in an array
array_search()	searches an array for a values and returns the index.
in_array()	Checks if a specified value exists in an array.
sort()	sort arrays in ascending order
rsort()	sort arrays in descending order
asort()	sort associative arrays in ascending order, according to the value
ksort()	sort associative arrays in ascending order, according to the key
arsort()	sort associative arrays in descending order, according to the value
krsort()	sort associative arrays in descending order, according to the key
array_intersect()	Compares array values and returns the matches.
array_merge()	Merges one or more arrays into one array
arra_push()	Inserts one or more elements at the end of an array
array_pop()	Deletes the last element of an array
array_shift()	Removes the first element from an array and reyrrns the the value of the deleted element.
array_sum()	Returns the sum of the values in an array
array_product()	Calculates the product of the values in an array
array_reverse()	Returns an array in reverse order
array_unique()	Removes duplicate values of a an array
array_values()	Retuens all the values of an array

<!-- Numeric Arrays -->

```
<html>
<body>
<?php
$cities = array("hyd" , "nzb", "wngl" , "vjk");
print_r($cities);
echo "<br>";
$res = print_r($cities , true);
echo "The result is : $res";
echo "<BR>";

echo "the third city is " . $cities[2];
echo "<br>";

// displaying the numeric array

echo "The array elements are <br> ";

foreach($cities as $str)
{
    echo "<h5 style='color:red' > $str </h2> ";
    echo "<br>";
}

$cities1[0]="hyd";
$cities1[1]="nzb";
$cities1[2]="wngl";
$cities1[3]="vjk";

echo "<BR>";
echo "the second city is " . $cities1[1];
echo "<BR>";


$name="vardhaman";
print_r($name); //any variable can be displayed.

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

<!-- Associative arrays -->

```
<html>
<body>
<?php

$sal = array("teja"=>35000 , "ravi"=>60000,"sahithi"=>45000);
print_r($sal);
echo "<br>";
echo 'The Ravi sal is '.$sal["ravi"];
echo "<br>";
print_r($sal);

// displaying the associative array

echo "The array elements are <br> ";

foreach($sal as $key => $val)
{
    echo "<h5 style='color:red' > $key : $val </h2> ";
    echo "<br>";
}

$sal1["teja"] =35000;
$sal1["ravi"] =60000;
$sal1["sahithi"] =45000;
print_r($sal);
echo "<br>";
echo 'The sahithi sal is '.$sal1["sahithi"];

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

<!-- Multi Dimensional Arrays -->

```
<html>
<body>
<?php

//Associative array
$emps= array( "developer" => array("ramesh" => 87000 , "suresh" =>97000 ,
"ganesh" => 99000),
"designer" => array("rajesh" => 165000 , "aneesh" =>115000 ,
"sathish" =>123000),
"tester"   => array("mahesh" => 45000 , "harish"=>35000));
```

```
print_r($emps);
echo "<br>";
echo "The testers salaries are " .print_r($emps["tester"],true) ;

echo "<br>";

// Numeric array + Associative array
$emps1= array( array("ramesh" => 87000 , "suresh" =>97000 , "ganesh" => 99000),
               array("rajesh" => 165000 , "aneesh" =>115000 , "sathish" =>123000),
               array("mahesh" => 45000 , "harish"=>35000));

print_r($emps1);
echo "<br>";
echo "The designer salaries are " .print_r($emps1[1],true) ;
```

// displaying the Multi dimensional array

```
echo "The array elements are <br> ";

foreach($emps1 as $categ)
{
    foreach($categ as $key=>$val)
    {
        echo "<div style='color:red' > $key : $val </div> ";
    }
    echo "<br>";

}
```

//numeric array -2D

```
echo "<br>";
$emps2= array( array( 87000 , 97000 , 99000),
               array(165000 , 115000 ,123000),
               array( 45000 , 35000));

print_r($emps2);
echo "<br>";
echo "The developer salaries are " .print_r($emps2[0],true) ;

foreach ($emps2 as $x)
    foreach($x as $y)
        echo "<br> $y ";
    ?>
</body>
</html>
```

<!-- Built in Array Functions -->

```
<html>
<body>
<?php
    $city = array("hyd" , "nzb", "wngl" , "vjk", "vzg" , "krnl");
    echo " the array is <br>";
    print_r($city);
    echo "<br>";
    //counting elements
    $len = count($city);
    echo "Array length is $len" ;
    echo "<br>";
    //searching for a element
    $index = array_search("wngl" , $city);
    echo "index of Warangal is $index";
    echo "<br>";
    //check a element exist or not
    $n = in_array("wngl",$city); //returns 1 if exist
    if($n==1)
        echo "<br> The element exist";
    else
        exit;
    //array intersect - compares two arrays and returns common
    $aa = array(12,13,67,89,34);
    $bb = array(15,87,34,13,60);
    $cc = array_intersect($aa,$bb);
    echo "<br> The common elements are ";
    print_r($cc);

    //merging arrays
    $dd = array_merge($aa, $bb);
    echo "<br> The Merge arrays are ";
    print_r($dd);

    //inserting one or more elements at end - push
    $p1=20;
    $p2=30;
    array_push($aa,$p1,$p2); //only variables pushed
    echo "<br> The array after push ";
    print_r($aa);

    //deleting last element -pop
    array_pop($aa);
    echo "<br> The array after pop ";
    print_r($aa);
```

```

//removes first element and returns -shift
$x =array_shift($aa);
echo "<br> The element deleted is $x ";
echo "<br> The array after delete ";
print_r($aa);

//The sum of array elements
$y = array_sum($aa);
echo "<br> The element sum  is $y ";

//The product of array elements
$z = array_product($aa);
echo "<br> The elements product is $z";

// reverse list of array is
echo "<br>";
echo "The reverse list is <br>";
print_r(array_reverse($city));

//displaying unique elements in an array
$list = Array(20,30,40,30,50,20);
echo "<br>";
echo "The list initially <br>";
print_r($list);
echo "<br> The unique elements are <br>";
print_r(array_unique($list));
//returns all the values
echo "<br>";
$m = array_values($list);
print_r($m);

?>
</body>
</html>

```

<!-- Array sort function in PHP -->

```

<html>
<body>
<?php
$city = array("hyd", "nzb", "wngl", "vjk", "vzg", "krnl");

//sorting elements - numeric array
sort($city);
echo "After sorting the list is <br>";
print_r($city);
echo "<br>";

```

```

// sorting the list - numeric array
$a = array(10,6,9,3,8,4);
sort($a);
print_r($a);
//reverse sort of elements
echo "<br> The reverse sort is ";
rsort($a);
print_r($a);

$sal = array("teja"=>35000 , "amar"=>25000,"ravi"=>60000,"sahithi"=>45000);
echo "<br> The list before sort <br>";
print_r($sal);
//Sorting the list- Associative array by value
echo "<br> The list after sort by value <br>";
asort($sal);
print_r($sal);
//Sorting the list- Associative array by Key
echo "<br> The list after sort by Key <br>";
ksort($sal);
print_r($sal);
//Reverse Sorting the list- Associative array by value
echo "<br> The list after reverse sort by value <br>";
arsort($sal);
print_r($sal);
//Reverse Sorting the list- Associative array by Key
echo "<br> The list after reverse sort by Key <br>";
krsort($sal);
print_r($sal);

?>
</body>
</html>
<!-- Array of differnt values -->

```

```

<html>
<body>
<?php
$cities = array("hyd" , 38, 42.56,20);
print_r($cities);
echo "<br>";

print_r($cities );

?>
</body>
</html>

```

Built-in String manipulation Functions

- Strings are used to represent most of the textual data.
- String is a collection of characters.

Function	Description
strlen()	Returns the length of the string. <code>int strlen(String \$var)</code>
explode()	Breaks a string into an array. <code>array explode(separator, string);</code>
Implode()	Join array elements with a string. <code>string implode(separator , array);</code>
strpos()	Finds the position of first occurrence of a string in another string, it is case sensitive. <code>int strpos(string , findstring , start);</code>
 strrchr()	Find the position of last occurrence of a string inside another string. <code>int strrchr(string , findstring)</code>
str_repeat()	Repeats a string for the specified number of times. <code>string str_repeat(string , times);</code>
strrev()	Reverses a string. <code>string strrev(string \$var);</code>
echo()	Outputs a string <code>echo(String);</code>
fprintf()	Writes a formatted string to specified output stream.
print()	Outputs a string
printf()	Outputs a formatted string.
Ltrim()	Deletes white spaces from left side of a string
rtrim()	Deletes white spaces from right side of a string.
trim()	Deletes white spaces from both sides of a string.
strtolower()	Converts a string to lowercase letters. <code>string strtolower(string \$var);</code>
strtoupper()	Converts a string to uppercase letters.

	String strtoupper(string \$var);
ucfirst()	Converts the first char of string to upper case. String ucfirst(string \$var);
ucwords()	Converts the first char of all words in a string to uppercase. String ucwords(string \$var);
str_word_count()	Counts the number of words in a given string. int str_word_count(string \$var)
str_split()	Splits a string into an array of specified no.of chararcters. array str_split(string , noofchars);
str_replace()	Replaces some characters in a string , case sensitive. string str_replace(oldstr, newstr, string \$var);
str_ireplace()	Replaces some characters in a string , case sensitive. string str_ireplace(oldstr, newstr, string \$var);
strcmp()	Compares to strings, case sensitive. int strcmp(string1 , strin2); This function returns: <ul style="list-style-type: none"> • 0 - if the two strings are equal • <0 - if string1 is less than string2 (-1) • >0 - if string1 is greater than string2 (1)
strcasecmp()	Compares to strings, case sensitive. int strcasecmp(string1 , strin2); This function returns: <ul style="list-style-type: none"> • 0 - if the two strings are equal • <0 - if string1 is less than string2 (ASCII value difference) • >0 - if string1 is greater than string2 (ASCII value difference)
strstr()	Search for a given pattern in the string, Retuns the rest of the string after match. string strstr(string \$var , string \$pattern).

<!-- String Manipulation Functions -->

```
<html>
<body>
<?php
    $str1 = "welcome";
    print_r($str1);
    // length of the string
    $len = strlen($str1);
    echo "<br> The length of string is $len <br>";
    //split string into an array
    $str2 = "well done ramesh";
    $list = explode(" " , $str2);
    echo "<br> The list after split with space is <br>";
    print_r($list);

    $str3 = "well#done#ramesh";
    $list = explode("#" , $str3);
    echo "<br> The list after split with # is <br>";
    print_r($list);
    //join array elements as a string
    $arr1 = Array("Ramesh" , "CSE" , 1093);
    $str4 = implode("#",$arr1);
    echo"<br> The array after join is <br>";
    echo $str4;
    $str4 = implode(" ",$arr1);
    echo"<br> The array after join is <br>";
    echo $str4;
    //searching for a string in string
    $str = "welcome to hyderabad ,soonhyderabad welcomes every one";
    $x = strstr($str , "hyd");
    echo "<br> The first occurence of string found at $x <br>";

    $y=strrpos($str , "bad");
    echo "<br> The last occurence of string found at $y <br>";
    //repeating a string n times
    $s1 ="wel";
    $s2 = str_repeat($s1 , 5);
    echo "<br> The repeated string is $s2 <br>";
    //reversing a string
    $s2 = "hyderabad";
    $s2= strrev($s2);
    echo "<br> The reversed string is ";
    echo("$s2"); //echo function
    echo "<br>";

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

<!-- String Functions -->

```
<html>
<body>
<?php
    $s1 = "welCoME";
    echo "The lower case string is " . strtolower($s1)."<br>";
    echo "The upper case string is " . strtoupper($s1)."<br>";
    echo "The first char is to upper case" . ucfirst($s1)."<br>";
    $s2="welcome to hyderabad";
    echo "The first char of every word to upper case" . ucwords($s2)."<br>";
    echo "The no of words in string are" . str_word_count($s2)."<br>";

    //split an array based on specified no of chars
    $arr1 = str_split($s1,2);
    print_r($arr1);

    //replace characters
    $s3 ="welcome to hyderabad";
    $s3=str_ireplace("hyderabad","secenderabad" , $s3);
    echo "<br> The string after replace $s3 <br>";

    //comparing two strings
    $ss1="hyderabad";
    $ss2="patna";
    $x1 = strcasecmp($ss1, $ss2);
    echo "<br> the comparison result is $x1 <br>";

    //find the pattern
    $s4 ="hello world is big";
    $y1= strstr($s4 , "world");
    echo "<br> $y1" ;

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

<!-- String Functions -->

```
<html>
<body>
<?php
    $x = 56;
    $y= 89;
    $z = "hello";
    printf("%15d , %8f , %s",$x,$y,$z);
    ?>
</body>
</html>
```

Mathematical Functions in PHP

- The mathematical functions can handle values within the range of integer and float types.

Function	Description
rand()	Generates a random integer. int rand(void) //returns a random integer between 0 and RAND_MAX. int rand(int min , int max)
log()	Returns the natural logarithm. Log(\$arg) arg -> the value to calculate the algorithm
round()	Rounds the float number. round(float \$val , int \$precision) val-> the value to round precision -> optional, the number of digits after the decimal point.
ceil()	Returns the rounded value of a given number. ceil(\$val)
floor()	Returns the value of a number rounded downwards to the nearest integer. floor(\$val)
abs()	Returns the absolute value of a number. abs(\$val)
exp()	Returns the value of e power x. exp(\$val)
fmod()	Returns the remainder of the division. fmod(\$val1 , \$val2)
pow()	Returns the value of x to the power y. pow(\$x ,&y)
sqrt()	Returns the square root of a number. sqrt(\$val)

max()	Returns the number with highest value of two specified numbers. max(\$x, \$y)
min()	Returns the number with lowest value of two specified numbers. Min(\$x, \$y)
pi()	Returns the value of PI. pi()
sin()	Returns the sine of a number. sin(\$val)
cos()	Returns the cosine of a number. cos(\$val)
tan()	Returns the tangent of a number. tan(\$val)
is_infinite()	Returns true if a value is an infinite number. is_infinite(\$val)
is_nan()	Return true if a value is an infinite number. is_nan()
base_convert()	Converts a number from one base to another. base_convert(\$val , base1 , base2)
bindec()	Converts a binary number to decimal. bindec(\$binval)
decbin()	Converts a decimal value to a binary. decbin(\$decval)
dechex()	Converts a decimal value to a Hexadecimal. Dechex(\$decval)
hexdec()	Converts a hexadecimal value to decimal. decbin(\$hexval)
octdec()	Converts a octal value to decimal. octdec(\$octval)
decoct()	Converts a decimal value to octal. octdec(\$decval)

<!--Math function using PHP-->

```
<html>
<body>
<?php

    //Randome Numbers
    echo(rand());
    echo "<br>";
    echo(rand());
    echo "<br>";
    echo(rand(10,20));
    echo "<br>";

    //Logarithmic values
    echo(log(2.5));
    echo "<br>";
    echo(log(0));
    echo "<br>";

    //rounding float numbers
    echo(round(34.56));
    echo "<br>";//34a
    echo(round(34.56666,3));
    echo "<br>";//34.566

    //ceil function
    echo(ceil(8.96));
    echo "<br>";//9

    //floor function
    echo(floor(8.76));
    echo "<br>";//8

    echo(abs(-8.96));
    echo "<br>";

    echo(exp(8.96));
    echo "<br>";

    echo(fmod(11,3));
    echo "<br>";

    echo(pow(8,3));
    echo "<br>";

    echo(sqrt(256));
    echo "<br>";
```

```
echo(max(25,86));
echo "<br>";

echo(min(25,86));
echo "<br>";

echo(pi());
echo "<br>";

echo(sin(10));
echo "<br>";

echo(cos(10));
echo "<br>";

echo(tan(90));
echo "<br>";

//conversion functions
$val = "EE23";
echo(base_convert($val,16,10)); //hexa to decimal
echo "<br>";

echo(bindec("1110"));
echo "<br>";

echo(decbin("14"));
echo "<br>";

echo(dechex("28"));
echo "<br>";

echo(hexdec("EF3"));
echo "<br>";

echo(octdec("56"));
echo "<br>";

echo(decoct("56"));
echo "<br>";

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

Working with forms

Data
Page

- The `<form>` tag in HTML is used to create a HTML form for user input.
- A form can contain elements like buttons, text fields, dropdown list (select), checkbox, radio buttons etc.
- The `<input>` element is used to add user controls to the form.

Eg : CSE `<input type="checkbox" name="c1" value="cse" >`
`</form>`.

Processing a Webform

The information entered in a webform is sent to a web server for processing.

The processing of webform include

1. Submitting the form data
2. Retrieving the form data
3. Validating the form data.

1. Submitting form data

The form data is submitted to the server for processing in two methods.

1. Using the get() method

When 'submit' button is pressed on web form,

- ① get() method sends the form data to the server by attaching it at the end of URL. (In Address bar).
- ② The attached form data is called query-string.
- ③ The get() method allows to send only a limited amount of information at a time.
- ④ The information sent by using the get() method is displayed on the address bar of the web browser.
- ⑤ Hence This approach is not used for sending passwords or any sensitive information.

2. Using the post() method

when 'submit' button is pressed on the form,

- ① post() method appends all the data in form into HTTP header message body.
- ② Used to send any amount of data.
i.e. Used to submit large and complex web forms.

- ③ Information sent through this method is not displayed in the URL of web browser.

2. Retrieving the Form Data

① $\$_GET[]$ function

- The function $\$_GET[]$ is a built in function used to retrieve values from a form sent through `get()` method.
- It is only for retrieving data sent through form's get method.

② $\$_POST[]$ function

- The function $\$_POST[]$ is a built in function used to retrieve values of a form sent through the `post()` method.

Eg: ① <?php echo $\$_GET["name"]$; ?>

② <?php echo $\$_POST["Name"]$; ?>

③. `$_REQUEST[]` function

Used to collect form data sent with the `get()` and `post()` methods.

E: `<?php echo $_REQUEST['name']; ?>`

④ `$ SERVER['REQUEST_METHOD']`

which method (get or post) used by the form to send data to the server is known from this `SERVER` method.

```
<?php  
if ($SERVER['REQUEST_METHOD']=='get')  
    echo "get method";  
else  
    echo "post method";  
?>
```

= =

3. Validating a form

- Form validation involves verifying that the user has entered the correct data in relevant form elements.
- The validating form on server side is more secure.

Eg: Checking for username Should not be empty

```
if (!$_POST['uname'])  
    echo "Enter valid user name.  
else
```

```
    echo " Credentials are correct.
```

1. Submitting data to server from a Web form

<!-- form submission using get method -->

```
<html>
<body>
<form name="f1" method="get" action="getread.php">

    User Name : <input type="text" name="uname"> <br>
    Mobile No : <input type="text" name="no"> <br>
    Department :
    <Select name="s1">
        <option value="CSE" > CSE </option>
        <option value="ECE" > ECE </option>
        <option value="EEE" > EEE </option>
    </select>
    <input type="submit" name="send" value="SUBMIT">
</form>
</body>
</html>
```

2. Retrieving data from the Web form

<!-- retrieving data from form using \$_GET -->

```
<html>
<body>

    <br>
    <br>
    Welcome : <?php echo $_GET["uname"]; ?>
    <br>
    Your Number is : <?php echo $_GET["no"]; ?>

    <br>
    Branch selected : <?php echo $_GET["s1"]; ?>

</body>
</html>
```

<!-- form submission using post method -->

```
<html>
<body>
<form name="f1" method="post" action="getpost.php">

    User Name : <input type="text" name="uname">      <br>
    Mobile No : <input type="text" name="no">          <br>
    Department :
    <Select name="s1">
        <option value="CSE" > CSE </option>
        <option value="ECE" > ECE </option>
        <option value="EEE" > EEE </option>
    </select>
    <input type="submit" name="send" value="SUBMIT">
</form>
</body>
</html>
```

<!-- retrieving data from form using \$_POST method -->

```
<html>
<body>
    Welcome : <?php echo $_POST["uname"]; ?>  <br>
    Your Number is : <?php echo $_POST["no"]; ?>  <br>
    Branch selected : <?php echo $_POST["s1"]; ?>

</body>
</html>
```

<!-- retrieving data from form using \$_REQUEST for both get and post requests -->

```
<html>
<body>
    <br>
    <br>
    Welcome : <?php echo $_REQUEST["uname"]; ?>  <br>
    Your Number is : <?php echo $_REQUEST["no"]; ?>  <br>
    Branch selected : <?php echo $_REQUEST["s1"]; ?>

</body>
</html>
```

3. Validating a form at server side

<!-- Server validating form -->

```
<html>
<body>
<?php

$flag=true;
$msg ="";

if($_POST['uname'])
{
    if(preg_match("/[^A-Z a-z]/" , $_POST['uname']))
    {
        $msg = (" Eneter Valid User Name")."<br>";
        $flag=false;
    }
}

if(!$_POST['pwd']) //checks for empty
{
    $msg = $msg.( " Enter valid password");
    $flag = false;
}

if($flag==false)
{
    //goback to complete form data correctly.
    echo $msg."<BR>". "<input type='button' value='back' onClick='history.go(-1)'>";
}
else
{
    echo "Correct data";
    //check with database.
}

?>

</body>
</html>
```

Date in PHP

date — Format a local time/date

date (string \$format))

date (string \$format [, int \$timestamp = time()])

Returns a string formatted according to the given format string using the given integer timestamp or the current time if no timestamp is given. In other words, timestamp is optional and defaults to the value of time().

Format:

format	character Description	Example returned values
	Day ---	---
d	Day of the month, 2 digits with leading zeros	01 to 31
D	A textual representation of a day, three letters	Mon through Sun
j	Day of the month without leading zeros	1 to 31
l(lowercase 'L')	A full textual representation of the day of the week	Sunday through Saturday
N	numeric representation of the day of the week	1 (for Monday) through 7 (for Sunday)
S	English ordinal suffix for the day of the month, 2 characters	st, nd, rd or th. Works well with j
w	Numeric representation of the day of the week	0 (for Sunday) through 6 (for Saturday)
z	The day of the year (starting from 0)	0 through 365
	Week ---	---
W	week number of year, weeks starting on Monday	Example: 42 (the 42nd week in the year)
	Month ---	---
F	A full textual representation of a month, such as January or March	January through December
m	Numeric representation of a month, with leading zeros	01 through 12
M	A short textual representation of a month, three letters	Jan through Dec
n	Numeric representation of a month, without leading zeros	1 through 12
t	Number of days in the given month	28 through 31
	Year ---	---
L	Whether it's a leap year	1 if it is a leap year, 0 otherwise.
Y	A full numeric representation of a year, 4 digits	Examples: 1999 or 2003
y	A two digit representation of a year	Examples: 99 or 03
	Time ---	---
a	Lowercase Ante meridiem and Post meridiem	am or pm
A	Uppercase Ante meridiem and Post meridiem	AM or PM
g	12-hour format of an hour without leading zeros	1 through 12
G	24-hour format of an hour without leading zeros	0 through 23

format	character Description	Example returned values
<i>h</i>	12-hour format of an hour with leading zeros	01 through 12
<i>H</i>	24-hour format of an hour with leading zeros	00 through 23
<i>i</i>	Minutes with leading zeros	00 to 59
<i>s</i>	Seconds with leading zeros	00 through 59
<i>Full Date/Time</i>		---
<i>c</i>	ISO 8601 date (added in PHP 5)	2004-02-12T15:19:21+00:00
<i>r</i>	» RFC 2822 formatted date	Example: <i>Thu, 21 Dec 2000 16:01:07 +0200</i>
<i>U</i>	Seconds since January 1 1970 00:00:00 GMT	

time — Return current Unix timestamp

time (void)

Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

<!-- Date in PHP -->

```
<html>
<body>
<?php

$today = date("F j, Y, g:i A");
echo"<br>1 $today";
$today = date("m.d.y");
echo"<br>2 $today";

$today = date("Y/m/d");
echo"<br>3 $today";

$today = date('h-i-s, j-m-y, it is w Day');
echo"<br>4 $today";

$today = date('\i\t \i\s \t\h\e jS \d\l\a\y.');
echo"<br>5 $today";

$today = date("D M j G:i:s T Y");
echo"<br>6 $today";
$today = date('H:m:s');
echo"<br>7 $today";
$today = date("H:i:s");
echo"<br>8$today";
$today = date("Y-m-d H:i:s");
```

```
echo"<br>9 $today";
$today = date("z");
echo"<br>10 $today";
$today = date("W");
echo"<br>11 $today";
$today = date("t");
echo"<br>12 $today";
$today = date("L"); //returns 1 if leap year else returns 0.
echo"<br>13 $today";
$today = date("c");
echo"<br>14 $today";
$today = date("r");
echo"<br>15 $today";
$today = date("U");
echo"<br>16 $today";

$nextWeek = time() + (7 * 24 * 60 * 60);

echo 'Now:    '.date('Y-m-d')."\n";
echo "<br>";
echo 'Next Week: '.date('Y-m-d', $nextWeek)."\n"; //using time step.
// or using strtotime():

echo "<br>";
echo "timeis".time(); //milliseconds

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

Working with Databases

Date _____
Page _____

- MySql is a RDBMS used by the world's largest and fastest growing organizations.
 - MySql is owned, developed & supported by Sun Microsystems.
 - One of the world's largest open source S/W contributors.
- * PHP is used to create dynamic web pages, which require a database to store and retrieve data.
- * The most widely used database in PHP is MySql.
- To access MySql database server, we need to check the configuration by

```
<?php  
    phpinfo();  
?>
```

Shows MySql Configuration in PHP.

Connecting to Database

In PHP, a database can be used by establishing a connection to the MySQL database server.

The information required to connect to database

- ① hostname (localhost | ipaddress)
- ② database username (usually - root)
- ③ password (usually "" - empty)
- ④ database name. (optional).

" If connection is required in multiple pages,
the 'connect' code is stored in a file and
imported like a normal file. i.e.

why? →
< ?php
 include 'filename'; // calls the file
?>

Connecting to MySQL Server

The functions for connecting to server are

`mysql_connect ("hostname", "username", "password")`
(or)

`mysqli_connect ("host", "username", "password", "databaseName")`

`host` - optional, specifies the host name or IP address. (for local server, it is "localhost").

`username` - optional, it specifies mysql password.

`password` - optional, it specifies mysql password

`databaseName` - optional, it is database name where operation performed on data.

The function returns an object which represent MySQL connection. If connection failed it returns `false`.

```
<?php  
$con = mysqli_connect ('localhost', 'root', '', 'shipment');  
if (! $con)  
    die ('Error in Connection'. mysqli_error());  
else  
    echo "Connection to Server was successful";  
    to obtain error message
```

≡

?>

Creating Database using PHP Script

The functions for creating a database is

`mysql_query ('query for create', connection)`
(or)

`mysqli_query (connection, query)`.

Eg:

```
$query = "CREATE DATABASE mydb";  
mysql_query ($con, $query);  
echo "Database created".
```

`mysqli_query()` function returns the result from database if it is successful.

If the function fails return false.

The `mysqli_query()` function is used to execute a query from PHP script.

Selecting the Database

The functions `mysql_select_db()`, `mysqli_select_db()` used to select the required database.

~~`mysql_select_db (dbname, conn)`~~

`mysqli_select_db (conn, dbname)`.

Returns true if the selection is successful
else return false.

Eg: \$sel = mysqli_select_db(\$con, \$dbname);
if (\$sel)
 echo "Database selected successfully";
else
 echo "Database not selected";

Creating a table in Database

```
$qry1 = "CREATE TABLE login  
(uname varchar(40) NOT NULL,  
pword varchar(20) NOT NULL)";
```

```
$tab = mysqli_query($con, $qry1);  
if ($tab)  
    echo "<br> Table created";  
else  
    echo "<br> Table not created";
```

— — —

Inserting Data into table

```
$qry = "INSERT INTO login(uname, pword)  
VALUES ('$uname', '$pword')";
```

```
E: mysqli_query($con, $qry)
```

Retrieving Data from a table

Table name ↗

```
$res = mysqli_query($con, "select * from login");
```

```
echo "<table border=2>";
```

```
echo "<tr> <th> USER NAME </th>
```

```
<th> Password </th> </tr>";
```

```
while ($row = mysqli_fetch_array($res))
```

```
{
```

```
echo "<tr> <td>";
```

```
echo $row['uname'];
```

```
echo "</td>";
```

```
echo "<td>";
```

```
echo $row['pword'];
```

```
echo "</td> </tr>";
```

```
}
```

```
echo "</table> ;
```

Database ↗

Table fields ↗

Updating a record in a table

```
$oldvalue = $_POST['old']; } Getting  
$newvalue = $_POST['new']; date from  
form  
mysql_query($con, "UPDATE login1 SET  
pword = '$newvalue' WHERE  
pword = '$oldvalue'"); Table name  
fieldname
```

deleting a record from a Table

```
$name = $_POST['user']; } Reading  
from form  
mysql_query($con, "DELETE FROM login1 WHERE  
uname = '$name'");
```

=

Creating Database Application using PHP

<!-- Creating Database through PHP Script -->

```
<html>
<body>
<?php

    $con = mysqli_connect('localhost','root','');
    if(!$con)
    {
        die('Error in Connection'.mysqli_error());
    }
    else
    {
        echo "Connection to server was successufull";
    }

    $dbname = "mydb";
    $qry = "CREATE DATABASE $dbname";
    mysqli_query($con , $qry);
    echo "<br> Database Created with name $dbname";

    $sel = mysqli_select_db($con , $dbname);
    if($sel)
        echo "<br> Database $dbname is selected successfully";
    else
        echo " <br> Database $dbname not selected";

    $qry1 = "CREATE TABLE login1 ( uname varchar(40) NOT NULL , pword varchar(20) NOT NULL)";

    $tab = mysqli_query($con , $qry1);
    if($tab)
        echo "<br> Login Table created";
    else
        echo "<br> Login Table Not created";

    mysqli_close($con);

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

<!-- Login Details Entry Form -->

```
<!-- Login Details Entry Form -->
<html>
<body>
<form name="f2" method="post" action="insertuser.php">
    <center>
        <table border="2" >
            <caption style="color:red"> Register Window </caption>
            <tr>
                <td> User Name :</td>
                <td> <input type="text" name="un" size=30 required="required"></td>
            </tr>
            <tr>
                <td> Password : </td>
                <td> <input type="password" name="pw" size=30 required="required"> </td>
            </tr>

            <tr>
                <td align="left" >
                    <input type="Submit" value="Insert User Details"> &nbsp;&nbsp;
                    <td> <a href="displayusers.php"> Display Users </a>
                <td style="color:red" href="verifyuser.php"> &nbsp;Existing User Verify </a> </td>
                </td>
            </tr>
            <tr>
                <td> <a href="updateform.php"> Update Password </td>
                <td> <a href="deleteform.php"> Delete User </a> </td>
            </tr>
            <tr>
                <td align="right"> <input type="reset" value="Clear"> </td>

                <td> </td>
            </tr>

        </table>
    </center>
</form>
</body>
</html>
```

<!-- Inserting Users Data : Code for insertuser.php -->

```
<html>
<body>
<?php

$con = mysqli_connect('localhost','root','','mydb');
if(!$con)
{
    die('Error in Connection'.mysqli_error());
}

$sel = mysqli_select_db($con , 'mydb');
if(!$sel)
{
    echo "<br> Database not selected";
}

$name = $_POST['un'];
$pwd = $_POST['pw'];

$qry2 = "INSERT INTO login1(uname,pword) VALUES('$name' , '$pwd')";

$ins = mysqli_query($con , $qry2);

if(!$ins)
    echo "<br> Login data Not inserted successfully";

echo "<br> User $name data inserted successfully";

echo"<br> <a href='login.php'>Move to Home Page </a>";

mysqli_close($con);

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

<!--Retriving Users - Displaying users data -->

```
<html>
<body>
<?php

    $con = mysqli_connect('localhost','root','','mydb');
    if(!$con)
    {
        die('Error in Connection'.mysqli_error());
    }

    $sel = mysqli_select_db($con , 'mydb');
    if(!$sel)
        echo " <br> Database not selected";

    $res = mysqli_query($con , "SELECT * From login1");

    echo "<center>";
    echo "<table border=2>";
    echo "<caption> Users Details </caption>";
    echo "<tr> <th> USER NAME </th> <th> PASSWORD </th> </tr> ";

    while($row = mysqli_fetch_array($res))
    {
        echo "<tr> <td> ";
        echo $row['uname'];
        echo "</td>";
        echo " <td> ";
        echo $row['pword'];
        echo "</td></tr>";
    }

    echo "</table>";
    echo "</ceneter>";

    echo"<br> <a href='login.php'>Move to Home Page </a>";
    mysqli_close($con);
?>

</body>
</html>
```

```
<!-- Update a field password -->
<html>
<body>
<form name="f2" method="post" action="update.php">
<center>
<table border="2" >
<caption style="color:red"> Register Window </caption>
<tr>
<td> User Name :</td>
<td> <input type="text" name="unn" size=30 required="required"></td>
</tr>
<tr>
<td> Old Password : </td>
<td> <input type="password" name="opw" size=30 required="required">
</td>
</tr>

<tr>
<td> New Password : </td>
<td> <input type="password" name="npw" size=30 required="required">
</td>
</tr>
<tr>
<td> </td>
<td align="left" >
<input type="Submit" value="UPDATE"> &nbsp;&nbsp;
<input type="reset" value="Clear">
</td>
</tr>
</table>

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

```
<!-- Update a record from users data : Code for update.php -->
```

```
<html>
<body>
<?php

    $con = mysqli_connect('localhost','root','','mydb');
    if(!$con)
    {
        die('Error in Connection'.mysqli_error());
    }

    $sel = mysqli_select_db($con , 'mydb');
    if(!$sel)
        echo "<br> Database is selected successfully";

    //Getting form Data

    $un = $_POST['unn'];
    $old = $_POST['opw'];
    $ne = $_POST['npw'];

    mysqli_query($con , "UPDATE login1 SET pword='$ne' WHERE pword='$old'");

    echo "<br>Updated User Data Successfully";
    echo "<br> <a href='login.php'>Move to Home Page </a>";
    mysqli_close($con);

?>

</body>
</html>
```

<!-- delete a USER DATA Form -->

```
<html>
<body>
<form name="f2" method="post" action="deleteuser.php">
<center>
<table border="2" >
<caption style="color:red"> Register Window </caption>
<tr>
<td> User Name :</td>
<td> <input type="text" name="user" size=30 required="required"></td>
</tr>
<tr>
<td align="left" >
<input type="Submit" value="Delete" > &nbsp;&nbsp;
<input type="reset" value="Clear" >
</td>
</tr>
</table>
</center>
</form>
</body>
</html>
```

<!-- Delete a record from users data : Code for deleteuser.php -->

```
<html>
<body>
<?php

$con = mysqli_connect('localhost','root','','mydb');
if(!$con)
{
    die('Error in Connection'.mysqli_error());
}

$sel = mysqli_select_db($con , 'mydb');

if(!$sel)
    echo " <br> Database not selected";

$name = $_POST['user'];

mysqli_query($con , "DELETE FROM login1 WHERE uname='".$name"");

echo "<br> Deleted User Data Successfully ";

echo"<br> <a href='login.php'>Move to Home Page </a>";
mysqli_close($con);

?>

</body>
</html>
```

```
<!--Verify User form: Code for verifyuser.php -->
<html>
<body>
<form name="f2" method="post" action="verify.php">
    <center>
        <table border="2" >
            <caption style="color:red"> verify user </caption>
            <tr>
                <td> User Name :</td>
                <td> <input type="text" name="un" size=30
required="required"></td>
            </tr>
            <tr>
                <td> Password : </td>
                <td> <input type="password" name="pw" size=30
required="required"> </td>
            </tr>

            <tr>
                <td align="left" colspan=2>
                    <input type="Submit" value="verify me">
                </td>
            </tr>
        </table>
    </center>
</form>
</body>
</html>
```

<!-- verifying user in server - verify.php-->

```
<html>
<body>
<?php

    $name=$_POST['un'];
    $pwd=$_POST['pw'];

    $con = mysqli_connect('localhost','root','','mydb');
    if(!$con)
    {
        die('Error in Connection'.mysqli_error());
    }

    $sel = mysqli_select_db($con , 'mydb');
    if(!$sel)
        echo " <br> Database not selected";

    $res = mysqli_query($con , "SELECT * From login1");

    while($row = mysqli_fetch_array($res))
    {
        if(strcmp($row['uname'],$name)==0 && strcmp($row['pword'],$pwd)==0)
        {
            echo "Welcome $name";
            break;
        }
    }

    echo "<br> <a href='login.php'>Move to Home Page </a>";
    mysqli_close($con);
?>

</body>
</html>
```

Introduction to XML

Date
Page

XML - Extensible Markup language

"XML is a markup language based on simple, platform independent rules for processing and displaying textual information in a structured way."

Applications of XML

XML is used for various operations implementation

- Configuring information
- Publishing
- Electronic data interchange
- voice mail systems
- Remote method invocation (RMI)
- Object serialization
- Vector graphics
- XML is intended for transport or storage of data.

Every XML document contains XML elements and attributes associated with it.

XML Characteristics

- XML is standardized by W3C. - 1998
- XML is used to store and transport the data over the network.
- XML is platform independent language
- XML has mostly userdefined tags
- Every XML element/tag has must open tag and close tag.
- Every browser provides a XML parser to process XML document.
- Any Text editor that supports ASCII/Unicode character system used to write XML document
- Any XML document must be saved with file extension '.xml'.
- If a XML document follows rules, then it is said to be "wellformed XML document".
- Every XML document must contain a 'root' element.

HTML

- ① It is used to display or present data.
- ② Most of the tags are predefined.
- ③ It is case insensitive language.
- ④ It will not check for syntax or rules.
- ⑤ It does not preserve white spaces.
- ⑥ Some of the tags are not closed. i.e. both container and non container tags.

XML

- It is used to describe, store and transport data.
- Most of the tags are userdefined.
- It is case sensitive language.
- Checks for syntax and rules.
- It preserves the white spaces.
- All the tags must have open and close tag.

Features of XML | Advantages of XML

- XML is intended for transport or storage of data
- XML allows, user is able to define and use his own tags, called custom tags. It makes not require to used only predefined tags defined by proprietary vendors.
- XML is for data storage, HTML is for presentation
- XML allows userdefined tags, HTML uses only predefined tags.
- XML contains only data and does not contain any formating information.
- XML allows to create tags for any type of information like mathematical formula, employee data etc. These can be described using XML
- XML document needs to be validated using External tools like DTD, Schema.
- XML is not Vendor specific or Browser specific but HTML is browser dependant.
- The manipulation of XML document, sorting, searching, rendering (presenting) is done using XSL. (Extensible Style sheet language).
- The data can be extracted from database and can be used in morethan one application. Different applications perform different tasks on this data.

- XML is useful for exchanging data between different applications.
- XML document is human readable, we can edit by using simple text editors.
- XML document is language neutral, i.e. A Java program can generate XML document and this document can be parsed/processed in PHP.
- XML document has a tree structure. Hence complex data can be arranged systematically and can understood in simple manner.
- XML files are platform independent. Hence can work on any platform.

Comments in XML

<!--

Specify the comments in multiple lines
-->

Goals of XML

- Users must be able to define and use their own tags. This allows us to restrict the use of set of tags defined by Proprietary vendors.
- Allows user to build own tag library based on web requirement.
- Allows user to define formatting rules for the user defined tags.
- XML must support for storage or transport of data.

XML document

Ordinary XML document

Well-formed XML document

[Structure + Data]

[Rules + Structure + Data]

DTD

Schema

(Document Type Definition).

Example XML document

<?xml version = "1.0" encoding = "ISO-8859-1"?>

<productData>

Root element

<product>

<PID> P123 </PID>

<PNAME> SAMSUNG A20 </PNAME>

<DESCRIPTION> Good Display </DESCRIPTION>

<QTY> 5 </QTY>

</product>

<product>

<PID> P345 </PID>

<PNAME> OPPO </PNAME>

<DESCRIPTION> Selfie style </DESCRIPTION>

<QTY> 10 </QTY>

</product>

≡

=

</productData>

Run on browser as

"c://demo/products.xml"

(or)

View XML document.

VALID XML documents

- A XML document is valid, if it is validated against rules defined by DTD or Schema.
- Then the document is also called well-formed XML document.

A well-formed XML document must follow the following Rules.

- ① Must have a starting tag and closing tag
- ② XML tags are case sensitive
- ③ XML elements must be properly nested
- ④ XML document must have a root element and only one root element.
- ⑤ XML attribute values must be quoted. (Single / double)
- E: <EMP title="Mr" > Ramesh </EMP>
- ⑥ In XML white space is preserved.
<EMP> A Ramesh </EMP>
- ⑦ XML declaration must be at the beginning of a file. The attribute of declaration is in the order version, encoding, etc.
- ⑧ If XML declaration is done, version must be specified, all other attributes are optional.

- ⑨ XML elements can start with letter or '-'
But not any other
- ⑩ Element name cannot contain spaces.
- ⑪ Element name cannot contain ":"
- ⑫ Element name cannot start with "xml" in any case.
- ⑬ No space after '<' char, but can have space before '>' closing char.
- ⑭ To display special or ambiguous characters entities are used.

Escaping
delimiters { }
Characters.

< (<),
> (>), <
" ← ", &, '

Example XML element Names

Eg: <emp> ✓
 <-emp> ✓
 <xml|emp> X
 <emp-no> ✓
 <emp1-no> ✓
 <EMP> & <emp> differs.
 <emp:name> X
 <2emp> X

Nested Elements

- An XML element can contain other nested elements.

Eg: - <EMPNAME>
 <FIRST> Adavelli </FIRST>
 <MIDDLE> Ramesh </MIDDLE>
 </EMPNAME>

Empty Elements

- An element may have no nested element or content in it.

Eg: - <MIDDLENAME/> .

Escaping Delimiter Characters

& -	&
' -	'
" -	"
< -	<
> -	>

Eg: <DOB data = "< 12/07/2019 >"> c/DOB

XML Attributes

- Provides additional information about elements

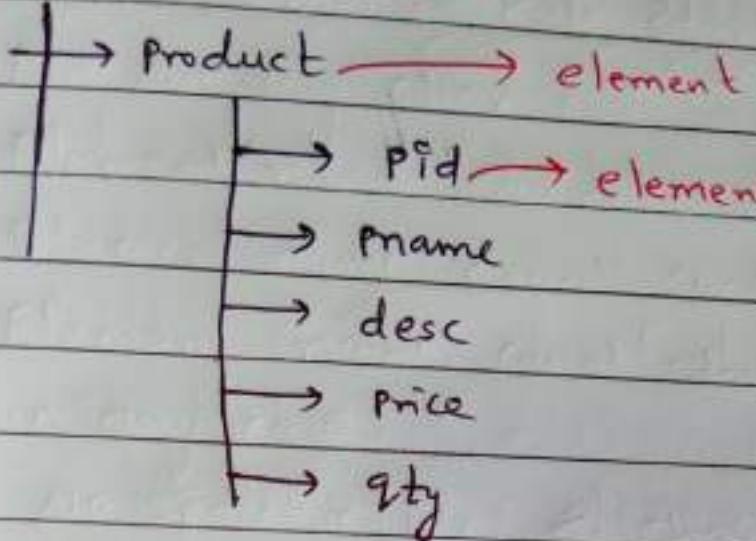
Eg: <EMPNO id = "VCE" > 1093 </EMPNO>
 <LASTNAME> Rddy </LASTNAME> .

Example XML Document

Date _____
Page _____

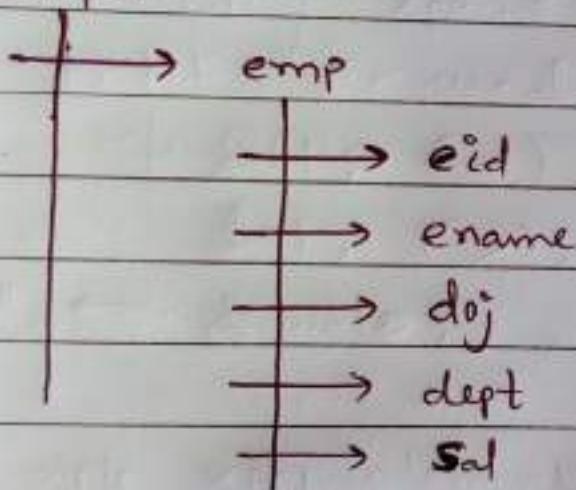
1

Products → root



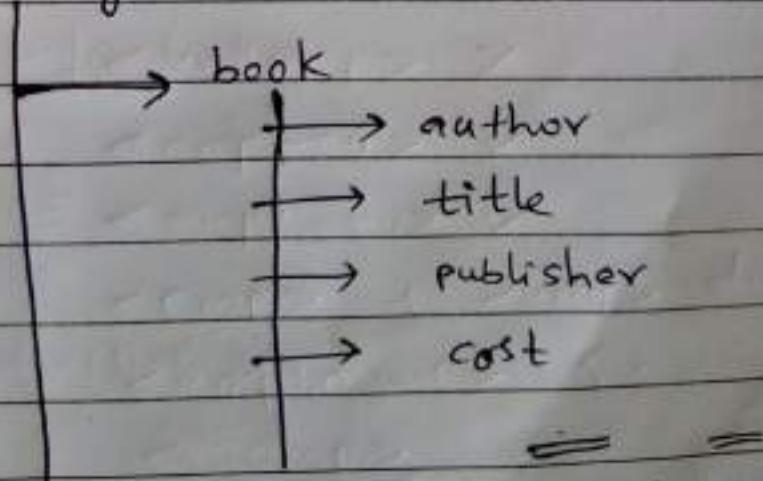
2

emps



3

catalog



//Example To Organize Employee Data

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

<emps>

    <emp>

        <eid> 1001 </eid>

        <ename> Varsha</ename>

        <doj> 12/10/2016 </doj>

        <dept> Developer </dept>

        <sal> 85000 </sal>

    </emp>

    <emp>

        <eid> 1002</eid>

        <ename>Harsha</ename>

        <doj> 1/5/2017</doj>

        <dept> Designer </dept>

        <sal> 65000 </sal>

    </emp>

    <emp>

        <eid> 1092</eid>

        <ename>Teja</ename>

        <doj> 11/05/2015</doj>

        <dept> Tester </dept>

        <sal> 55000 </sal>

    </emp>

</emps>
```

//Example To Organize Product Information

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

<PRODUCTS>

    <PRODUCT>

        <PID> S123 </PID>

            <PNAME>SAMSUNG A20 </PNAME>

            <DESC>This has High Quality Display </DESC>

            <PRICE>19000</PRICE>

            <QTY> 10 </QTY>

    </PRODUCT>

    <PRODUCT>

        <PID> S516 </PID>

            <PNAME>iPhone8</PNAME>

            <DESC>This will do all perfect</DESC>

            <PRICE>125000</PRICE>

            <QTY> 4 </QTY>

    </PRODUCT>

    <PRODUCT>

        <PID> S778 </PID>

            <PNAME>OPPO</PNAME>

            <DESC>Selfie Camera</DESC>

            <PRICE>25000</PRICE>

            <QTY> 8 </QTY>

    </PRODUCT>

</PRODUCTS>
```

Document Type Definition(DTD)

- DTD is used to define the rules and attributes for using the tags in a XML document.
- An XML document can be defined as:

Well-formed: If the XML document adheres to all the general XML rules such as tags must be properly nested, opening and closing tags must be balanced, and empty tags must end with '/>', then it is called as well-formed.

- A well-formed XML document is an XML document with correct syntax.
OR

Valid: An XML document said to be valid when it is not only well-formed, but it also conforms to available DTD that specifies which tags it uses, what attributes those tags can contain, and which tags can occur inside other tags, among other properties.

- If a XML document follows the rules given by the DTD then the document is **valid xml document**.
- A well formed XML document can be validated against DTD or Schema.
- A DTD defines the XML document structure with a list of legal elements and attributes.
- A DTD defines the structure of XML document like elements and their relation, hierarchy, attributes and entities.
- DTDs check the validity of structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.

A DTD describes (Features)

- ✓ The elements that can appear in an XML document.
- ✓ The order in which the elements can appear.
- ✓ Optional and mandatory elements.
- ✓ Element attributes and whether they are optional or mandatory.
- ✓ Whether attributes can have default values.

The set of rules for a XML document can be specified using **DTD Types** as

1. **Inline or Internal DTD:** The structure and rules of XML document are specified in the XML document.

- ✓ A DTD is referred to as an internal DTD if elements are declared within the XML files.
- ✓ To reference it as internal DTD, **standalone** attribute in XML declaration must be set to **yes**. This means the declaration works independent of external source.

The syntax of internal DTD is:

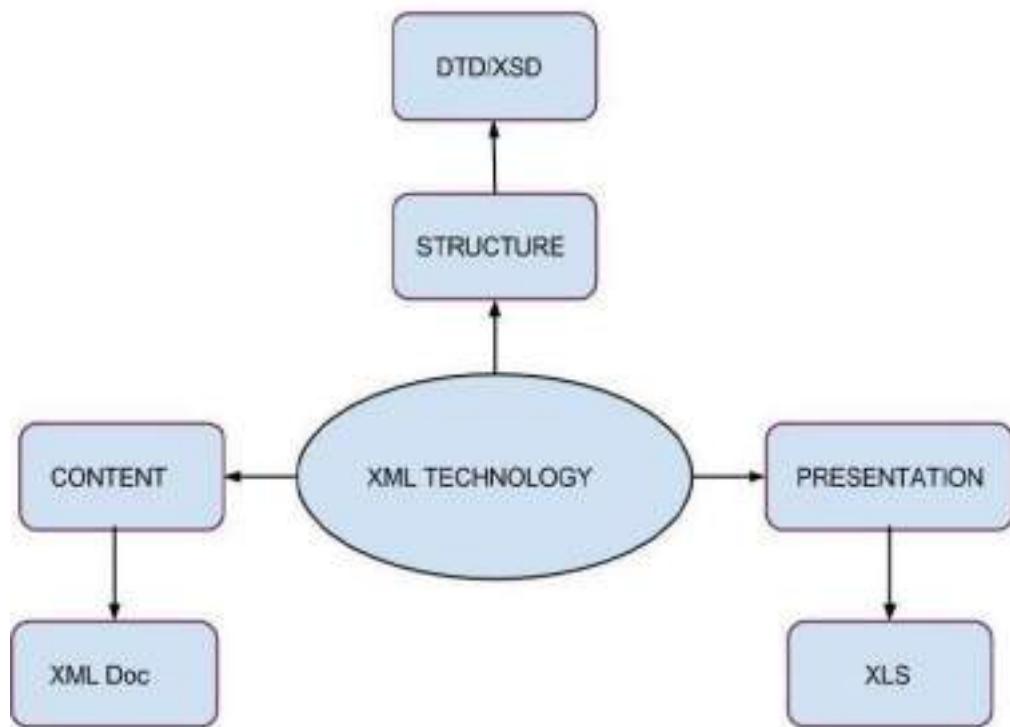
`<!DOCTYPE root-element [element-declarations]>`

2. **External DTD:** The structure and rules of XML document are specified in a separate file with “.dtd” extension, referenced in XML document.

- ✓ In external DTD elements are declared outside the XML file. They are accessed by specifying the **system** attributes which may be either the legal **.dtd** file or a **valid URL**.
- ✓ To reference it as external DTD, **standalone** attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Following is the syntax for external DTD:

`<!DOCTYPE root-element SYSTEM "file-name/URL">`



Advantages of using DTD

1. **Documentation** - We can define own format for the XML files. Looking at this document a user/developer can understand the structure of the data.
2. **Validation** - It gives a way to check the validity of XML files by checking whether the elements appear in the right order, mandatory elements and attributes are in place, the elements and attributes have not been inserted in an incorrect way, and so on.

Disadvantages of using DTD

1. It does not support the namespaces. Namespace is a mechanism by which element and attribute names can be assigned to groups. However, in a DTD namespaces have to be defined within the DTD, which violates the purpose of using namespaces.
2. It supports only the text string data type.(does not support datatypes).
3. It is not object oriented. Hence, the concept of inheritance cannot be applied on the DTDs.
4. Limited possibilities to express the cardinality for elements.
5. DTD will not use XML syntax, hence certain xml processors and xml frameworks do not understand DTD.
6. DTD will not use namespaces.

DTD Components / DTD Elements

1. Elements (`<!ELEMENT>`)
2. Attributes (`<!ATTLIST>`)
3. Entities (`<!ENTITY>`)

1. Defining Elements and Structure (`<!ELEMENT>`)

- ✓ XML elements can be defined as building blocks of an XML document. Elements can behave as a container to hold text, elements, attributes, media objects or mix of all.
- ✓ Each XML document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or empty elements.
- ✓ A DTD element is declared with an **ELEMENT** declaration. When an XML file is validated by DTD,
- ✓ Parser initially checks for the root element and then the child elements are validated.

Syntax : `<!ELEMENT elementname (content)>`

1. `<!ELEMENT elementname EMPTY >` //Empty Element

Eg: `<!ELEMENT address EMPTY>`

The XML element in XML file should be `<address/>`

2. `<!ELEMENT elementname e (child1, child2...)>` //Child Elements

Eg: `<!ELEMENT PRODUCT(PID,PNAME,DESC,PRICE,QTY)>`

3. `<!ELEMENT elementname (#PCDATA)>` //Element containing data

Eg: `<!ELEMENT PNAME (#PCDATA)>`

4. `<!ELEMENT elementname e (#PCDATA|child1|child2) >` //Mixed type element `<!ELEMENT address (#PCDATA|home | office)*>`

The Qualifiers for DTD definition are

Qualifier	Meaning
<code>?</code>	Optional , Zero or One occurrence <code><!ELEMENT address name?></code>
<code>*</code>	Zero or more Occurrences <code><!ELEMENT address name * ></code>
<code>+</code>	one or more Occurrences <code><!ELEMENT address name + ></code>
<code>,</code>	sequence of child elements separated by comma <code><!ELEMENT address (name, company, phone)></code>
<code> </code>	It allows making choices in the child element. <code><!ELEMENT address (home office)></code>

```

<!ELEMENT PRODUCTS (PRODUCT+)> //root element
<!ELEMENT PRODUCT(PID,PNAME,DESC,PRICE,QTY)> //nested elements
<!ELEMENT PID(#PCDATA)> //element containing data
<!ELEMENT PNAME(#PCDATA)>
<!ELEMENT DESC(#PCDATA)>
<!ELEMENT PRICE(#PCDATA)>
<!ELEMENT QTY(#PCDATA)>

```

2. Defining DTD Attributes (<!ATTLIST>)

- Attribute gives more information about an element or more precisely it defines a property of an element.
- An XML attribute is always in the form of a name-value pair.
- An element can have any number of unique attributes.
- We declare a list of allowable attributes for each element ,
- All attributes are defined using **ATTLIST** declaration.

Basic syntax of DTD attributes declaration is:

<!ATTLIST element-name attribute-name attribute-type attribute-value>

We want specify that an attribute is required.

Eg: <!ATTLIST name id CDATA #REQUIRED>

<!ATTLIST element-name attribute-name attribute-type "default-value">

Eg: <!ATTLIST name id CDATA "0">

<!ATTLIST element-name attribute-name attribute-type #FIXED "value" >

#FIXED keyword followed by the fixed value is used when you want to specify that the attribute value is constant and cannot be changed. A common use of fixed attributes is specifying version numbers.

Eg:

<!ATTLIST company name #FIXED "INFOSYS">

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

If the attribute you are declaring has no default value, has no fixed value, and is not required, then we must declare that the attribute as **implied**. Keyword **#IMPLIED** is used to specify an attribute as implied.

Eg: <!ATTLIST name id CDATA #IMPLIED>

PCDATA -> The PCDATA text is parsed by The XML parser.

(Parsed Character Data)

CDATA -> The text inside CDATA is ignored by the XML Parser. Unparsed character data. (A text string).

-> indicates the string followed by it is a special keyword not xml element name.

enumeration -> Set of Enumerated values.

3. Defining DTD Entities (!ENTITY)

- ✓ Entities are used to substitute a value for an XML element.
- ✓ Entities are a mechanism to define replacement values
- ✓ XML supports User defined entities or Predefined entities (<).
- ✓ Entities can be declared internally or externally

Internal Entity

If an entity is declared within a DTD it is called as internal entity.

Following is the syntax for internal entity declaration:

`<!ENTITY entity_name "entity_value">`

Eg: `<!ENTITY phone_no "(011) 123-4567">`

```
<address>
    &phone_no;
</address>
```

External Entity

If an entity is declared outside a DTD it is called as external entity. You can refer to an external Entity by either using system identifiers or public identifiers.

Following is the syntax for External Entity declaration:

`<!ENTITY name SYSTEM "URI/URL">`

```
<!DOCTYPE books SYSTEM "books.dtd" [
    <!ENTITY copyright SYSTEM "copy.xml"> //external
    <!ENTITY call-us "123456789"> ] > //internal0
```

Inline or Internal DTD Examples

```
<!--Employees Info Using Inline or Internal DTD -->
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE emps [
    <!ELEMENT emps (emp+)>
    <!ELEMENT emp (id,name,sal,dept,company)>
    <!ELEMENT id (#PCDATA)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT sal (#PCDATA)>
    <!ELEMENT dept (#PCDATA)>
    <!ELEMENT company (#PCDATA)>
    <!ATTLIST dept grade CDATA #REQUIRED
        block CDATA "A">
    <!ATTLIST id prefix CDATA #IMPLIED>
    <!ENTITY address "Infosys Technoligies, Gachibowli, Hyderabad">
]>

<emps>
    <emp>
        <id prefix="INF">101</id>
        <name>Ramesh</name>
        <sal>85900</sal>
        <dept grade="B"> Developer</dept>
        <company>&lt;address;&gt;</company>
    </emp>
    <emp>
        <id>121</id>
        <name>Suresh</name>
        <sal>65900</sal>
        <dept grade="C"> Tester</dept>
        <company>&quot;address;&quot;</company>
    </emp>
    <emp>
        <id>131</id>
        <name>Ganesh</name>
        <sal>185900</sal>
        <dept grade="A">Designer</dept>
        <company>&apos;address;&apos;</company>
    </emp>
```

```
</emps>
<!--Products Info Using Inline or Internal DTD →
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE products [
    <!ELEMENT products (product+)>
    <!ELEMENT product (pid,pname,desc,price,qty)>
    <!ELEMENT pid (#PCDATA)>
    <!ELEMENT pname (#PCDATA)>
    <!ELEMENT desc (#PCDATA)>
    <!ELEMENT price (#PCDATA)>
    <!ELEMENT qty (#PCDATA)>
]>

<products>
<product>
<pid> LG102 </pid>
    <pname> LG ULTRA HD 50 </pname>
    <desc> High Definition </desc>
    <price> 119000 </price>
    <qty> 12 </qty>
</product>

<product>
<pid> SS110 </pid>
    <pname> SAMSUNG CURVE HD 50 </pname>
    <desc> High UV Definition </desc>
    <price> 139000 </price>
    <qty> 8 </qty>
</product>

<product>
<pid> MI222 </pid>
    <pname> MI CURVE HD 50 </pname>
    <desc> High UV Definition </desc>
    <price> 69000 </price>
    <qty> 14 </qty>
</product>

</products>
```

External DTD Examples

<!—Employees Info Using External DTD →

emp.dtd

```
<!ELEMENT emps (emp+)>
<!ELEMENT emp (id,name,sal,dept,company)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT sal (#PCDATA)>
<!ELEMENT dept (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ATTLIST dept grade CDATA #REQUIRED
      block CDATA "A">
<!ATTLIST id prefix CDATA #IMPLIED>
<!ENTITY address "Infosys Technoligies, Gachibowli, Hyderabad">
```

empexter.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE emps SYSTEM "emp.dtd" >
<emps>
  <emp>
    <id prefix="INF">101</id>
    <name>Ramesh</name>
    <sal>85900</sal>
    <dept grade="B"> Developer</dept>
    <company>&lt;&address;&gt;</company>
  </emp>
  <emp>
    <id>121</id>
    <name>Suresh</name>
    <sal>65900</sal>
    <dept grade="C"> Tester</dept>
    <company>&quot;&address;&quot;</company>
  </emp>
  <emp>
    <id>131</id>
    <name>Ganesh</name>
    <sal>185900</sal>
    <dept grade="A">Designer</dept>
    <company>&apos;&address;&apos;</company>
  </emp>
```

```
</emps>
<!--Products Info Using External DTD →
      product.dtd
<!ELEMENT products (product+)>
<!ELEMENT product (pid,pname,desc,price,qty)>
<!ELEMENT pid (#PCDATA)>
<!ELEMENT pname (#PCDATA)>
<!ELEMENT desc (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT qty (#PCDATA)>
```

[prodexter.xml](#)

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE products SYSTEM "product.dtd" >
<products>
  <product>
    <pid> LG102 </pid>
    <pname> LG ULTRA HD 50 </pname>
    <desc> High Definition </desc>
    <price> 119000 </price>
    <qty> 12 </qty>
  </product>

  <product>
    <pid> SS110 </pid>
    <pname> SAMSUNG CURVE HD 50 </pname>
    <desc> High UV Definition </desc>
    <price> 139000 </price>
    <qty> 8 </qty>
  </product>

  <product>
    <pid> MI222 </pid>
    <pname> MI CURVE HD 50 </pname>
    <desc> High UV Definition </desc>
    <price> 69000 </price>
    <qty> 14 </qty>
  </product>
</products>
```

<!—MotorBikes Info Using internal DTD →
motorbike -> make,model,year,color,engine,chasis-num,accessories.
engine -> eng-no,cylinders,fuel-type.
accessories attributes -> disk-brake => “yes or no” , auto-start =>“yes or no”.
use appropriate entities.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE catalog [
    <!ELEMENT catalog (motorbike*)>
    <!ELEMENT motorbike (make,model,year,color,engine,chasisnum,accessories)>
    <!ELEMENT make (#PCDATA)>
    <!ELEMENT model (#PCDATA)>
    <!ELEMENT year (#PCDATA)>
    <!ELEMENT color (#PCDATA)>
    <!ELEMENT engine (eng-no,cylinders,fueltype)>
    <!ELEMENT eng-no (#PCDATA)>
    <!ELEMENT cylinders (#PCDATA)>
    <!ELEMENT fueltype (#PCDATA)>
    <!ELEMENT chasisnum (#PCDATA)>
    <!ELEMENT accessories (#PCDATA)>
    <!ATTLIST accessories diskbrake (yes|no) #REQUIRED>
    <!ATTLIST accessories autostart (yes|no) #REQUIRED>
    <!ENTITY mirror "TwoSides">
]>
```

```
<catalog>
    <motorbike>
        <make> HeroHonda</make>
        <model>PassionPro</model>
        <year> 2019</year>
        <color>Black</color>
        <engine>
            <eng-no> 2345EF</eng-no>
            <cylinders>2 </cylinders>
            <fueltype> Petrol</fueltype>
```

```
</engine>
<chasisnum>1122</chasisnum>
<accessories diskbrake="yes" autostart="no"> ALL Reuired , &mirror;</accessories>
</motorbike>

<motorbike>
<make> Honda Active</make>
<model>5G</model>
<year> 2018</year>
<color>RED</color>
<engine>
<eng-no> 12345EF</eng-no>
<cylinders> 4 </cylinders>
<fueltype> Petrol</fueltype>
</engine>
<chasisnum>110022</chasisnum>
<accessories diskbrake="yes" autostart="yes"> ALL Reuired , &mirror;</accessories>
</motorbike>
</catalog>
```

XML Namespaces

- It is possible to have two document types have elements with same name, but different meanings and semantics.
- Using XML namespaces we can differentiate elements and attributes of different XML document types from each other when combining them together into other documents or processing multiple documents simultaneously.
- XML Namespaces provide a method to avoid element name conflicts.
- In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

Eg:

```
<student>
    <BTECH>
        <course> Computer Science </course>
        <age> 22 </age>
    </BTECH>
    <MTECH>
        <course> DataScience </course>
        <age> 24 </age>
    </MTECH>
</student>
```

Here would be a name conflict for `<course>` and `<age>`, but the elements have different content and meaning.

Name conflicts in XML can easily be avoided using a **name prefix**.

```
<student xmlns:ug="http://www.w3.org/2001/XMLSchema"
         xmlns:pg="http://www.w3.org/2001/XMLSchema" >
    <BTECH>
        <ug:course> Computer Science </ug:course>
        <ug:age> 22 </ug:age>
    </BTECH>
    <MTECH>
        <pg:course> DataScience </pg:course>
        <pg:age> 24 </pg:age>
    </MTECH>
</student>
```

XML Schema

- XML Schema is used to represent the structure of a XML document.
- XML schema is used to define the building blocks of a XML document.
- The XML schema language is also called XML Schema Definition (XSD) language.
- It is recommended by W3C in 2001.
- An XML Schema describes the structure of an XML document, just like a DTD.
- An XML document with correct syntax is called "**Well Formed**".
- An XML document validated against an XML Schema is both "**Well formed**" and "**Valid**".
- XML Schema defines elements, attributes, child elements .
- It also defines fixed and default values of elements and attribute.
- Allows the developer to use **data types**.

Advantages / Features of XML Schemas

- XML Schemas are written in XML
- XML Schemas use XML Syntax
- XML Schemas are extensible to additions
- XML Schemas support data types
 - It is easier to describe document content
 - It is easier to define restrictions on data
 - It is easier to validate the correctness of data
 - It is easier to convert data between different data types
- XML Schemas support namespaces
- With XML Schema, your XML files can carry a description of its own format.
- With XML Schema, independent groups of people can agree on a standard for interchanging data.
- With XML Schema, we can verify data.
- No need to learn a new language
- We can use XML editor to edit Schema files
- We can use XML parser to parse Schema files
- We can manipulate Schemas with the XML DOM
- We can transform Schemas with XSLT.

XML Schema (XSD) Elements

XSD elements are classified into two types.

1. Simple Types

- ✓ An element contains only a value and it does not contain child elements or attributes.
- ✓ Simple type element is used only in the context of the text.
- ✓ The simple type element is created using <xs:simpleType>.

Eg: <age> 30 </age>

```
<xs:simpleType name="age" type="xs:integer"/>
```

2. Complex Types

- ✓ An element contains child elements and attributes.
- ✓ 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.
- ✓ The complex type element is created using <xs:complexType>.

Eg: <address>

```
    <name> Infosys </name>
    <company> Hyderabad </company>
    <phone> 1234056789 </phone>
</address>
```

```
<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:integer" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Schema (XSD) Data types

The data types supported in schemas for XML elements are

1. string data type

The “string” data type is used to define the element containing characters, lines or white spaces. The element can have initial or default values , fixed values.

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

The element in XML document must contain
<company> Infosys </company>

2. numeric data type

The numeric values to element can be defined using “integer” or “decimal”.

```
<xs:element name="ElementName" type="xs:integer"/>
<xs:element name=ElementName" type="xs:decimal"/>
```

<xs:element name="age" type="xs:integer"/>
The element in XML document must contain
<age> 25 </age>

<xs:element name="avg" type="xs:decimal"/>
The element in XML document must contain
<avg> 85.95 </avg>

3. date data type

The date in an XML element is define using “date” data type.
The date format is in the form yyyy-mm-dd, all the three entries must present.

```
<xs:element name="ElementName" type="xs:date"/>
```

```
<xs:element name="dob" type="xs:date"/>
```

The element in XML document must contain
<dob> 2012-05-10 </dob>

4. time data type

The time for an XML element can be defined using “time” data type.

The time format is hh:mm:ss

```
<xs:element name="ElementName" type="xs:time"/>
```

```
<xs:element name="starttime" type="xs:time"/>
```

The element in XML document must contain

```
<starttime> 11:45:52 </starttime >
```

5. boolean data type

The true or false value can be assigned to xml element using “boolean”.

```
<xs:element name="ElementName" type="xs:boolean"/>
```

```
<xs:element name="flag" type="xs:boolean"/>
```

The element in XML document must contain

```
<flag> true </flag >
```

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 "HYD":

```
<xs:element name="city" type="xs:string" default="HYD"/>
```

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

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

```
<xs:element name="city" type="xs:string" fixed="HYD"/>
```

XML Schema (XSD) Indicators

The XML Schema uses various indicators to control how the elements can be used in XML document.

1. Order Indicators.

- Used to define the order of the XML elements in the document.

a) All indicator

- ✓ The `<all>` indicator specifies that the child elements can appear in any order, and that each child element must occur only once.
- ✓ When using the `<all>` indicator you can set the `<minOccurs>` indicator to 0 or 1 and the `<maxOccurs>` indicator can only be set to 1

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

b) Choice Indicator

- ✓ The `<choice>` indicator specifies that either one child element or another can occur

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="string"/>
      <xs:element name="member" type="string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

c) Sequence Indicator

- ✓ The `<sequence>` indicator specifies that the child elements must appear in a specific order.

```

<xs:element name="emp">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="eid" type="xs:string"/>
      <xs:element name="ename" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

2. Occurrence indicators.

- ✓ The Occurrence indicators are used to define how often an element can occur.
- ✓ The default value for maxOccurs and minOccurs is 1.

a) maxOccurs Indicator

- ✓ The <maxOccurs> indicator specifies the maximum number of times an element can occur.
- ✓ maxOccurs="unbounded" specifies any number of elements included.
- ✓ The meta characters
 - * => minOccurs=0 , maxOccurs="unbounded"
 - + => minOccurs=1 , maxOccurs="unbounded"
 - ? => minOccurs=0 , maxOccurs="1"
 -

b) minOccurs Indicator

- ✓ The <minOccurs> indicator specifies the minimum number of times an element can occur.

```

<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
                  maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

3. Group Indicators.

- ✓ Group indicators are used to define related sets of elements

a) Element groups

- ✓ Element groups are defined with the group declaration.

```
<xs:group name="groupname">
...
</xs:group>
Eg:
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:element name="person" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

b) Attribute groups

Attribute groups are defined with the attributeGroup declaration.

```
<xs:attributeGroup name="groupname">
...
</xs:attributeGroup>

<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="prefix" type="xs:string"/>
  <xs:attribute name="grade" type="xs:string"/>
</xs:attributeGroup>

<xs:element name="person">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

XML Schema (XSD) Restrictions

- ✓ Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.
- ✓ **<xs:restriction>** is used to specify the restrictions over data used for XML element.

Restrictions on Values-Specifying range for a element.

- ✓ It uses **base** attribute to specify the type of value.
- ✓ **minInclusive**, **maxInclusive** are used.

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="18"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions on a Set of Values

- ✓ limit the content of an XML element to a set of acceptable values,
- ✓ we would use the **enumeration** constraint.

```
<xs:element name="branch">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="CSE"/>
      <xs:enumeration value="IT"/>
      <xs:enumeration value="ECE"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions on a Series of Values

- ✓ To limit the content of an XML element to define a series of numbers or letters that can be used.
- ✓ We would use the **pattern** constraint.

```

<xs:element name="ename">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([A-Z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

```

Restrictions on Length

- ✓ To limit the length of a value in an element, we would use the **length**, **maxLength**, and **minLength** constraints.

An element "password" with a restriction. The value must be exactly 8 characters:

```

<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/> //exact no.of chars
    </xs:restriction>
  </xs:simpleType>
</xs:element>

```

An element "password" with a restriction. The value must be minimum 5 characters and maximum 8 characters:

```

<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/> //minimum no.of chars
      <xs:maxLength value="8"/> //maximum no.of chars
    </xs:restriction>
  </xs:simpleType>
</xs:element>

```

XML Schema (XSD) Attributes

- ✓ All attributes are declared as simple types.
- ✓ Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.
- ✓ `<xs:attribute>` IS USED TO DEFINE ATTRIBUTES.
- ✓ The syntax for defining an attribute is:

```
<xs:attribute name="attributename" type="datatype"/>
```

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

The element in XML file should be assigned grade attribute as
`<student grade="first"> A.Ramesh </student>`

Default and Fixed Values for Attributes

- ✓ Attributes may have a default value OR a fixed value specified.
- ✓ A **default value** is automatically assigned to the attribute when no other value is specified.

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

- ✓ A **fixed value** is also automatically assigned to the attribute, and we cannot specify another value.

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

Optional and Required Attributes

- ✓ Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

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

Examples on XML Schema

//Example student schema rules and structure.

```
<?xml version="1.0"?>
<xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xselement name="student">
        <xsccomplexType>
            <xsssequence>
                <xselement name="name" type="xs:string"/>
                <xselement name="rollno" type="xs:string"/>
                <xselement name="course" type="xs:string"/>
                <xselement name="avg" type="xs:string"/>
            </xsssequence>
        </xsccomplexType>
    </xselement>
</xsschema>
```

//Example student schema XML file.

```
<?xml version="1.0" ?>
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="studsche.xsd">
    <name> A.Ramesh</name>
    <rollno>530</rollno>
    <course>CSE</course>
    <avg> 79</avg>
</student>
```

//Example book schema rules and structure.

```
<?xml version="1.0"?>
<xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xselement name="catalog">
        <xsccomplexType>
            <xsssequence>
                <xselement ref="book" minOccurs="1"
maxOccurs="unbounded" />
            </xsssequence>
        </xsccomplexType>
    </xselement>

    <xselement name="book">
        <xsccomplexType>
            <xsssequence>
                <xselement name="title" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xselement name="author" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xselement name="publisher" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xselement name="pages" type="xs:integer" minOccurs="1" maxOccurs="1"/>
                <xselement name="price" type="xs:decimal" minOccurs="1" maxOccurs="1"/>
            </xsssequence>
            <xselement name="course" type="xs:string" use="required"/>
        </xsccomplexType>
    </xselement>
</xsschema>
```

//Example book schema XML.

```
<?xml version="1.0" ?>
<!-- <xmstylesheet type="text/css" href="books.css"?> -->
<catalog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="books.xsd">
    <book course="cse" >
        <title>    Web Technologies </title>
        <author>Ramesh</author>
        <publisher>DreamTech</publisher>
        <pages>1200</pages>
        <price>450.50</price>
    </book>
```

```
<book course="cse">
    <title>      Operating Systems </title>
    <author>Galvin</author>
    <publisher>TMcH</publisher>
    <pages>900</pages>
    <price>850.50</price>
</book>
<book course="ece">
    <title>      Control Systems </title>
    <author>David Herny</author>
    <publisher>BPB</publisher>
    <pages>600</pages>
    <price>1250.70</price>
</book>
</catalog>
```

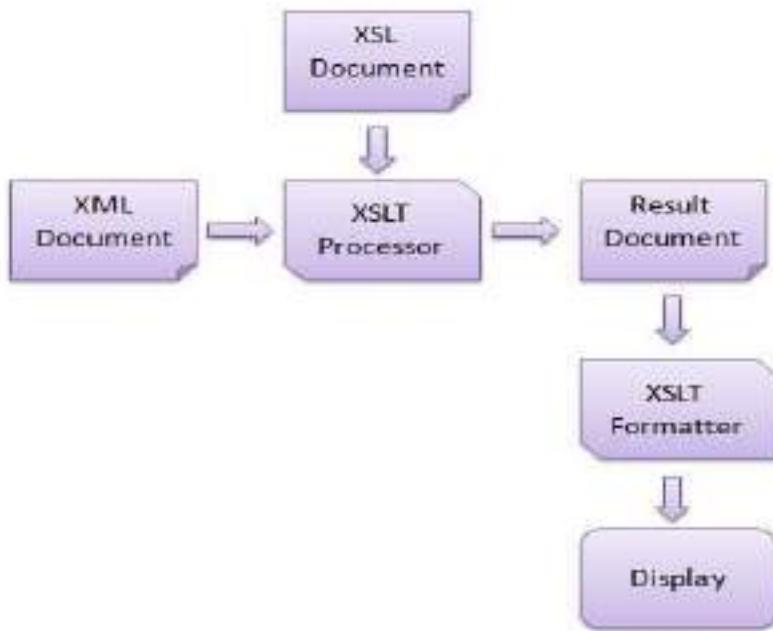
//Example book styles in css.

```
title {font-family:serif;color:green;font-size:18pt}
author{font-family:arial;color:red;font-size:12pt}
publisher{font-family:arial;color:blue;font-size:12pt}
pages{font-family:arial;color:red;font-size:12pt}
price{font-family:arial;color:blue;font-size:12pt}
```

XSLT

- **XSL (eXtensible Stylesheet Language)** is a styling language for XML.
- XSLT stands for XSL Transformations.
- XSLT is a language for transforming XML documents.
- EXtensible Stylesheet Language Transformation commonly known as XSLT is a way to transform the XML document into other formats such as XHTML.
- XSLT, Extensible Stylesheet Language Transformations, provides the ability to transform XML data from one format to another automatically.
- Automatic conversion of XML file to a HTML page

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



Advantages

Here are the advantages of using XSLT –

- Independent of programming. Transformations are written in a separate **xsl** file which is again an XML document.
- Output can be altered by simply modifying the transformations in **xsl** file. No need to change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.
- HTML tags Used for formatting purpose. XSLT Processor will skip them and browser will simply render them.
- **<xsl:stylesheet>** - xsl stylesheet declaration with xsl namespace: Namespace tells the xslt processor about which element is to be processed and which is used for output purpose only .

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

- **<xsl:template>** - template tells the xslt processor about the section of xml document which is to be formatted. It takes an XPath expression. In general, it is matching document root element and will tell processor to process the entire document with this template. defines a way to reuse templates in order to generate the desired output for nodes of a particular type/context.
<xsl:template match = "/">
- **<xsl:value-of>** tag puts the value of the selected node as per expression as text.
<xsl:value-of select = "firstname"/>
- **<xsl:for-each>** tag applies a template repeatedly for each node. processing instruction looks for each element matching the XPath expression
<xsl:for-each select = "class/student">
- **<xsl:sort>** tag specifies a sort criteria on the nodes.
<xsl:sort select = "firstname"/>

- **<xsl:if>** tag specifies a conditional test against the content of nodes.

```
<xsl:if test = "marks > 60">
```

- **<xsl:choose>** tag specifies a multiple conditional tests against the content of nodes in conjunction with the **<xsl:otherwise>** and **<xsl:when>** elements.

```
<xsl:choose>
  <xsl:when test = "marks >= 70">
    Distinction
  </xsl:when>
  <xsl:otherwise>
    Fail
  </xsl:otherwise>
</xsl:choose>
```

- **<xsl:key>** tag element specifies a named name-value pair assigned to a specific element in an XML document.

```
<xsl:for-each select = "key('firstname-search', 'ramesh')">
```

- **<message>** tag element helps to debug an XSLT processing. It is similar to javascript alerts. **<xsl:>** tag buffers a message to XSLT processor which terminates the processing and sends a message to the caller application to display the error message.

```
<xsl:if test = "firstname = ''">
  <xsl:message terminate = "yes">A first name field is empty.
  </xsl:message>
</xsl:if>
```

//Example to Present Book Catalog – book.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type = "text/xsl" href ="mystyle1.xsl" ?>
<catalog>
  <book>
    <title> XML for Web</title>
    <author>Winston</author>
    <publication>Pearson</publication>
```

```
<edition>8</edition>
<price>932Rs</price>
</book>

<book>
<title> XML for Web</title>
<author>Winston</author>
<publication>Pearson</publication>
<edition>8</edition>
<price>932Rs</price>
</book>
<book>
<title> XML for Web</title>
<author>Winston</author>
<publication>Pearson</publication>
<edition>8</edition>
<price>932Rs</price>
</book>
<book>
<title> XML for Web</title>
<author>Winston</author>
<publication>Pearson</publication>
<edition>8</edition>
<price>932Rs</price>
</book>
<book>
<title> XML for Web</title>
<author>Winston</author>
<publication>Pearson</publication>
<edition>8</edition>
<price>932Rs</price>
</book>
<catalog>
```

```
//mystyle1.xsl
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xsl:template match="/">
<html>
    <body>
        <table align="center" border="1">
            <caption> <h2>Books Information</h2></caption>
            <tr bgcolor="#BBEFFF">
                <th style="text-align:left">Title</th>
                <th style="text-align:left">Author</th>
                <th style="text-align:left">Publication</th>
                <th style="text-align:left">Edition</th>
                <th style="text-align:left">Price</th>
            </tr>
            <xsl:for-each select="catalog/book">
                <tr>
                    <td><xsl:value-of select="title"/></td>
                    <td><xsl:value-of select="author"/></td>
                    <td><xsl:value-of select="publication"/></td>
                    <td><xsl:value-of select="edition"/> </td>
                    <td><xsl:value-of select="price"/></td>
                </tr>
            </xsl:for-each>
        </table>
    </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

//Example to Present Student Information student.xml

```
<?xml version = "1.0"?>
<?xmlstylesheet type = "text/xsl" href = "studstyle.xsl"?>
<class>
  <student rollno = "561">
    <firstname>Rohith</firstname>
    <lastname>reddy</lastname>
    <marks>85</marks>
  </student>
  <student rollno = "573">
    <firstname>Vaneet</firstname>
    <lastname>Rao</lastname>
    <marks>55</marks>
  </student>
  <student rollno = "593">
    <firstname>ramesh</firstname>
    <lastname>kumar</lastname>
    <marks>64</marks>
  </student>
  <student rollno = "533">
    <firstname>Kushal</firstname>
    <lastname>swaroop</lastname>
    <marks>44</marks>
  </student>
</class>
```

```

//studstyle.xsl
<?xml version = "1.0" encoding = "UTF-8"?>

<xsl:stylesheet version = "1.0"
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:template match = "/">

<html>
    <body>
        <h2>Students</h2>

        <table border = "1">
            <tr bgcolor = "#9acd32">
                <th>Roll No</th>
                <th>First Name</th>
                <th>Last Name</th>
                <th>Marks</th>
            </tr>

            <xsl:for-each select="class/student">
                <tr>
                    <!-- @ is used to access attribute -->
                    <td><xsl:value-of select = "@rollno"/></td>
                    <td><xsl:value-of select = "firstname"/></td>
                    <td><xsl:value-of select = "lastname"/></td>
                    <td><xsl:value-of select = "marks"/></td>

                </tr>
            </xsl:for-each>

        </table>
    </body>
</html>
</xsl:template>
</xsl:stylesheet>

```

```
//sorting data based on particular element (sort)-stud.xsl
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0"
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:template match = "/">
    <html>
        <body>
            <h2>Students</h2>
            <table border = "1">
                <tr bgcolor = "#9acd32">
                    <th>Roll No</th>
                    <th>First Name</th>
                    <th>Last Name</th>
                    <th>Marks</th>
                </tr>

                <xsl:for-each select = "class/student">

                    <xsl:sort select = "firstname"/>
                    <tr>
                        <td><xsl:value-of select = "@rollno"/></td>
                        <td><xsl:value-of select = "firstname"/></td>
                        <td><xsl:value-of select = "lastname"/></td>
                        <td><xsl:value-of select = "marks"/></td>
                    </tr>
                </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

```
//selecting data based on condition (if)-stud.xsl
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0"
  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:template match = "/">
  <html>
    <body>
      <h2>Students</h2>
      <table border = "1">
        <tr bgcolor = "#9acd32">
          <th>Roll No</th>
          <th>First Name</th>
          <th>Last Name</th>
          <th>Marks</th>
        </tr>

        <xsl:for-each select = "class/student">

          <xsl:if test = "marks > 60">
            <tr>
              <td><xsl:value-of select = "@rollno"/></td>
              <td><xsl:value-of select = "firstname"/></td>
              <td><xsl:value-of select = "lastname"/></td>
              <td><xsl:value-of select = "marks"/></td>
            </tr>
          </xsl:if>
        </xsl:for-each>

      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

//using choose, when otherwise-studstyle2.xsl

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0"
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:template match = "/">
    <html>
        <body>
            <h2>Students</h2>
            <table border = "1">
                <tr bgcolor = "#9acd32">
                    <th>Roll No</th>
                    <th>First Name</th>
                    <th>Last Name</th>
                    <th>Marks</th>
                    <th>Grade</th>
                </tr>

                <xsl:for-each select = "class/student">

                    <tr>
                        <td><xsl:value-of select = "@rollno"/></td>
                        <td><xsl:value-of select = "firstname"/></td>
                        <td><xsl:value-of select = "lastname"/></td>
                        <td><xsl:value-of select = "marks"/></td>

                        <td>
                            <xsl:choose>
                                <xsl:when test = "marks >= 70">
                                    Distinction
                                </xsl:when>

                                <xsl:when test = "marks >= 60" >
                                    First
                                </xsl:when>
                            </xsl:choose>
                        </td>
                    </tr>
                <xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

```
<xsl:when test = "marks >= 50" >
    Second
</xsl:when>
<xsl:otherwise>
    Fail
</xsl:otherwise>
</xsl:choose>
</td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

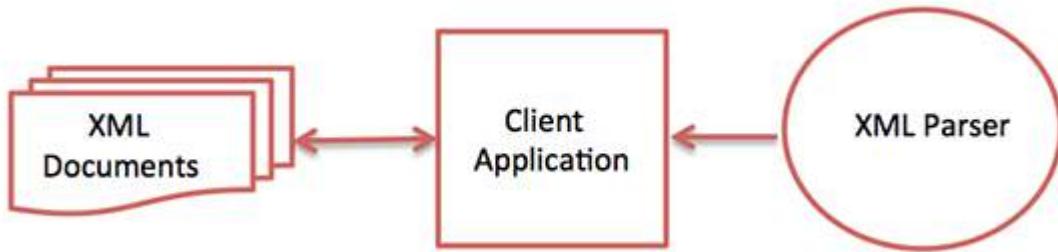
XML Parsers

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.

The goal of a parser is to transform XML into a readable code.



- ✓ When a software program reads an XML document and takes actions accordingly, this is called *processing* the XML.
- ✓ Any program that can read and process XML documents is known as an *XML processor*.
- ✓ An *XML processor* reads the XML file and turns it into in-memory structures that the rest of the program can access.
- ✓ The most fundamental XML processor reads an XML document and converts it into an internal representation for other programs or subroutines to use. This is called a *parser*, and it is an important component of every XML processing program.

XML Processor Types

Validating Parsers – Check XML documents for validity.

Eg: MSXML (Microsoft, in Java), Java Project X (Sun, in Java), xmlproc (Python)

Non Validating Parser-Not Check the XML document for validity.

OpenXML (Java), xmllib (Python).

A Parser is a program or software library used to check the given document is valid or not.

If it is valid, the XML document is converted into some memory representation to access the document content.

XML Parsers are in two classifications.

1. **DOM Parser (Document Object Model Parsers)**
2. **SAX Parsers. (Simple API for XML Parser).**

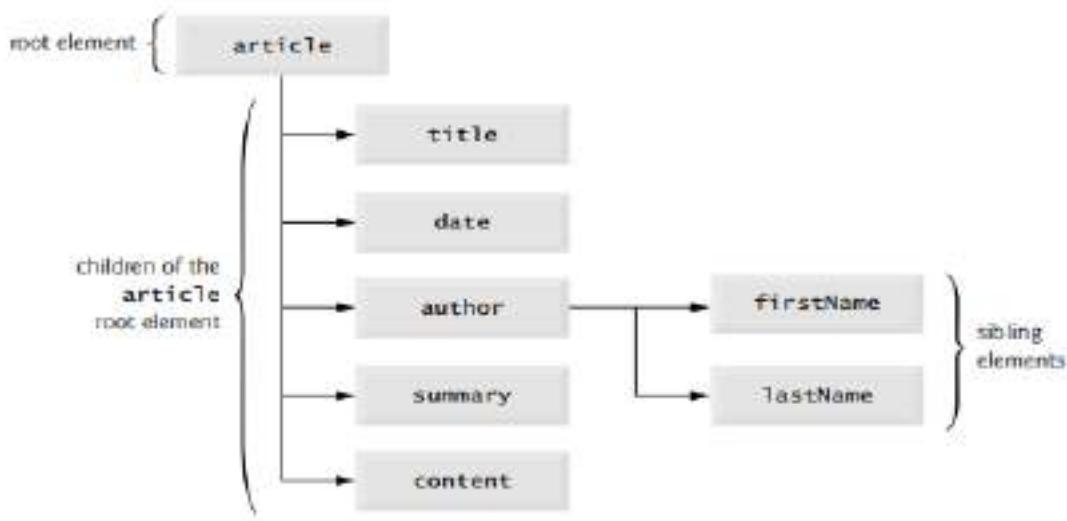
1. DOM Parser

- ✓ Upon successfully parsing a document, The XML parsers store document data as tree structures in memory.
- ✓ DOM parser load full XML file in memory and creates a tree representation of XML document.
- ✓ If sufficient amount of memory in server we can choose DOM as this faster because load entire xml in memory and works as tree structure which is faster to access.
- ✓ for small and medium sized XML documents, DOM is much faster

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

    <title> C Programming </title>
    <date> July 4 , 2001 </date>
    <author>
        <firstname> Dennies </firstname>
        <lastname> Ritchi </lastname>
    </author>
    <summary> C is Basic Programming </summary>
    <content> This book presents variables,functions , loops </content>
</article>
```

The tree structure for the root element of the document **article.xml**



- This hierarchical tree structure is called a Document Object Model (**DOM**) tree, and an XML parser that creates this type of structure is known as a DOM parser.

In DOM Parser

- ✓ The DOM tree has a single **root node**, which contains all the other nodes in the document.
- ✓ Each element name (e.g., article, date, firstName) is represented by a **node**.
- ✓ A node that contains other nodes (called **child nodes** or **children**) is called a **parent node** (e.g., author).
- ✓ A parent node can have many children, but a child node can have only one parent node.
- ✓ Nodes that are peers (e.g., firstName and lastName) are called **sibling nodes**.
- ✓ Uses the **XML DOM API** to display and manipulate the document's element names and values.
- ✓ The DOM tree contains various nodes like **root node**, **element node**, **attribute node**, **value node** etc.
- ✓ Tree structure allows traversing in top to bottom and vice versa.

The API used by DOM for handling XML document is

Property/Method	Description
Node (XML Element)	
nodeName	The name of the node.
nodeValue	A string or null depending on the node type.
parentNode	The parent node.
childNodes	A NodeList with all the children of the node.
firstChild	The first child in the Node's NodeList.
lastChild	The last child in the Node's NodeList.
attributes	A collection of Attr objects containing the attributes for this node
insertBefore	Inserts the node (passed as the first argument) before the existing node (passed as the second argument). If the new node is already in the tree, it's removed before insertion
NodeList	
item(index i)	Method that receives an index number and returns the element node at that index. Indices range from 0 to <i>length</i> – 1.
length()	The total number of nodes in the list.
Document (XML File)	
documentElement	The root node of the document.
createElement	Creates and returns an element node with the specified tag name.
createAttribute	Creates and returns an Attr node with the specified name and value.
getElementsByName	Returns a NodeList

Element	
tagName	The name of the element.
getAttribute	Returns the value of the specified attribute.
setAttribute	Changes the value of the attribute passed as the first argument to the value passed as the second argument.
removeAttribute	Removes the specified attribute.
Attribute	
value	The specified attribute's value.
name	The name of the attribute.

2. SAX Parser

- ✓ SAX (Simple API for XML) is an event-based parser for XML documents.
- ✓ Unlike a DOM parser, a SAX parser creates no parse tree. SAX is a streaming interface for XML, which means that applications using SAX receive event notifications about the XML document being processed an element, and attribute, at a time in sequential order starting at the top of the document, and ending with the closing of the ROOT element.

In SAX Parser

- ✓ Reads an XML document from top to bottom, recognizing the tokens that make up a well-formed XML document.
- ✓ Tokens are processed in the same order that they appear in the document.

- ✓ The application program provides an "event" handler that must be registered with the parser.

- ✓ As the tokens are identified, callback methods in the handler are invoked with the relevant information.

SAX Parser is used

- ✓ We can process the XML document in a linear fashion from top to down.
- ✓ The document is not deeply nested.
- ✓ When processing a very large XML document whose DOM tree would consume too much memory.
- ✓ The problem to be solved involves only a part of the XML document.
- ✓ Data is available as soon as it is seen by the parser, so SAX works well for an XML document that arrives over a stream.

Disadvantages of SAX

- ✓ We have no random access to an XML document since it is processed in a forward-only manner.
- ✓ If you need to keep track of data that the parser has seen or change the order of items, we must write the code and store the data on your own.

The API used by SAX for handling XML document is

`void startDocument()` – Called at the beginning of a document.

`void endDocument()` – Called at the end of a document.

`void startElement(String uri, String localName, String qName, Attributes atts)` – Called at the beginning of an element.

`void endElement(String uri, String localName, String qName)` – Called at the end of an element.

`void characters(char[] ch, int start, int length)` – Called when character data is encountered.

`void ignorableWhitespace(char[] ch, int start, int length)` – Called when a DTD is present and ignorable whitespace is encountered.

Attributes Interface

This interface specifies methods for processing the attributes connected to an element.

int getLength() – Returns number of attributes.

String getQName(int index) – returns name of attribute specified by index

String getValue(int index) - returns value of attribute specified by index

String getValue(String qname)- returns value of attribute specified by name

Differences between DOM and SAX parsers

DOM Parser	SAX Parser
DOM stands for Document Object Model	SAX stands for Simple API for XML Parsing
Tree Based Parser	Event Based Parser
Load entire document in memory as DOM Tree	Does not load entire document.
More Memory required storing entire tree structure.	Requires less memory
Effectively suitable for smaller and efficient memory, not for large XML files.	SAX is suitable for large XML files
DOM provides API for traversing the tree in top to bottom and vice versa and in random access.	SAX provides API for traversing only in top to bottom.
DOM allows to perform read, write and update XML document.	SAX does allow only read operation, cannot update directly.
The API for DOM is provided by the XML Parser.	The API for SAX is provided by the application languages like javascript, php, java.

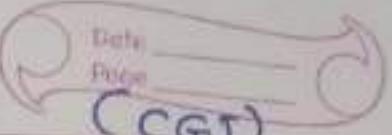
Web Servers & Servlets.

- Servlets are small programs that execute on the server side of a web connection.
- Servlets dynamically extend the functionality of a web server.
- Java Servlets are programs that run on a web or application server.
- Act as a interface layer between a request coming from a web browser or other HTTP client and database.

Using Servlets

- ① Collect input from users through web forms
- ② Present records from a database
- ③ Create web pages dynamically.

Common Gateway Interface (CGI)



- Before servlets CGI used as standard method for developing dynamic web applications.
- CGI is the interface used to communicate with web server.

Using CGI

- ① CGI allowed separate process to read data from HTTP request and write data to the HTTP response.
- ② for each of the client request a separate process is created. which is very complex to manage and heavy weight.
- ③ The languages used to build CGI programs are C, C++, Perl etc. These are not platform independent.
- ④ It was more expensive in terms of processor and memory resources to create separate process for each client. Thus suffer from serious performance problems.
- ⑤ Limit is made on no. of requests.

Using Servlets

- ① When a client makes a request, no separate process is created, instead a separate thread is created which runs within the address space of webserver.
- ② Servlets are platform independent. Bcz these are completely written in Java.
- ③ Java Security Manager on the Server enforces a set of restrictions to protect the resources of Server machine.
- ④ Performance is significantly better than CGI.
- ⑤ The full functionality of Java class libraries is available to a Servlet. i.e. A Servlet can communicate with applets, databases or other S/W via sockets.

Servlets Tasks

Servlets perform the tasks like

- ① Read the explicit data sent by the client
- ② Read the implicit HTTP request data sent by client like cookies.
- ③ Process the data and generate result.
- ④ Send the explicit data to the client
(HTML, XML, image etc)
- ⑤ Send the implicit HTTP response to the client like setting Cookie information, caching info etc.

Servlets - Life cycle

30th
Page

The three methods `init()`, `service()` & `destroy()` are central to the life cycle of servlet.

The life-cycle of Servlet is

- ① When user enters URL on browser, the browser then generates an HTTP Request for the URL. Then the request is sent to appropriate server.
- ② HTTP request is received by the web server, the Server maps the request to a particular servlet. The Servlet is dynamically retrieved and loaded into address space of the Server.
- ③ The Server invokes `init()` method of servlet. This method is invoked only once when the servlet is loaded first into memory. It is used for one time initialization, The initialization parameters are passed to configure the Servlet.
- ④ The Server invokes the `service()` method of the servlet. This method is called to process the HTTP Request.
"Reads Data provided by HTTP Request and formulate an HTTP response for client."

Each time a Server receives a request for a servlet, the server creates new thread and calls Service() method

The Service() method checks the request type (GET, PUT, POST, DELETE etc) and calls doGet(), doPost(), doPut() etc methods.

```
public void service ( ServletRequest req,  
                      ServletResponse res ) throws  
                      ServletException, IOException  
{  
    ==  
}  
}
```

A GET Request is handled by doGet() method

```
public void doGet ( HttpServletRequest req,  
                   HttpServletResponse res ) throws  
                   ServletException, IOException  
{  
    // Servlet code  
}
```

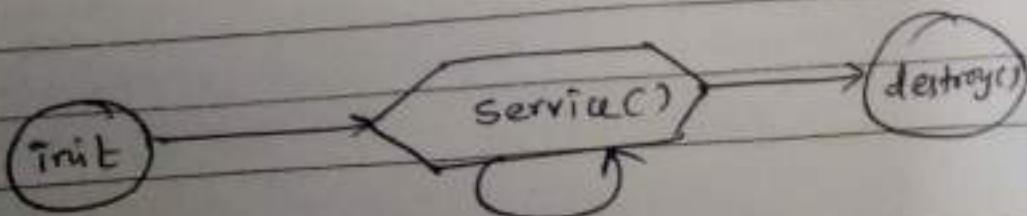
- ④ A POST request results from an HTML form
Can be handled using doPost().

```
public void doPost (HttpServletRequest req,  
                    HttpServletResponse res) throws  
{  
    ServletException, IOException  
    // Servlet code  
}
```

- The service() method is called for each service request
- The Servlet remains in server address space and is available to process any HTTP request.

- ⑤ The Server calls destroy() method to relinquish any resources such as file handlers, data source Pointers and saves any important data to storage before a servlet is unloaded from memory.

When a servlet is no longer accessed/needed, the memory allocated for the servlet and its objects can then garbage collected.



```
public void init() throws SeavletException  
{
```

// initialization code

{

```
public void destroy()
```

{

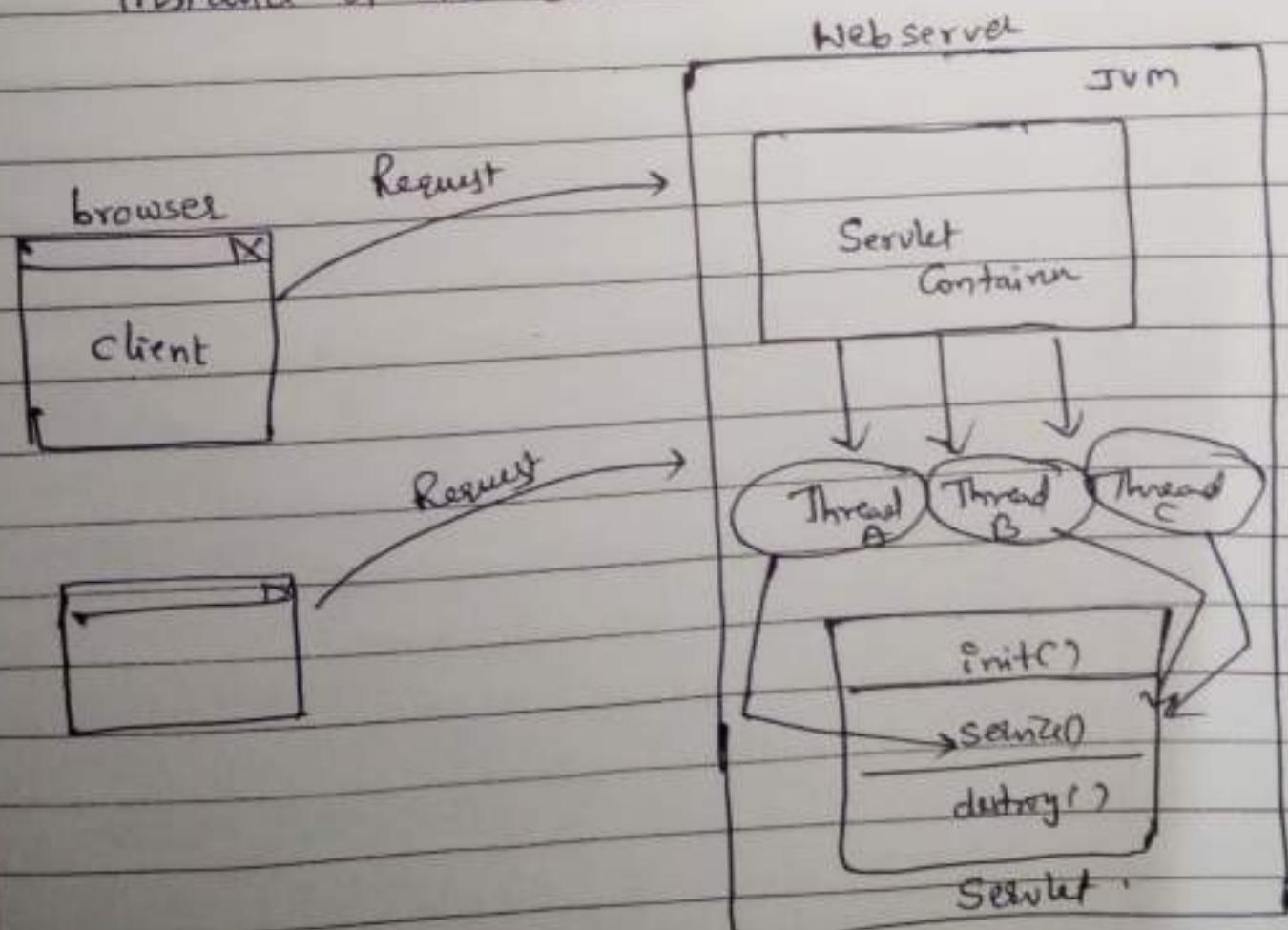
// finalization code

{



Servlet Architecture Diagram

- ① The HTTP request Coming to the Server are delegated to the Servlet Container
- ② The Servlet container loads the Servlet before invoking service() method
- ③ The Servlet Container handles multiple requests by spawning multiple threads , each thread executes the service() method of a single instance of the Servlet.



Servlet Skeleton

Date _____

Page _____

```
import java.io.*;
import javax.servlet.*;
public class ServletDemo extends GenericServlet
{
    public void init() throws ServletException
    {
        // initialization of all parameters
    }

    public void service(ServletRequest req,
                        ServletResponse res) throws
        ServletException, IOException
    {
        // Request processing
    }

    public void destroy()
    {
        // finalization code
    }
}
```

Web Servers

Date _____
Page _____

- Web server is a program that uses HTTP to serve files that create webpages to users in response to their requests.
- Used to create dynamic webpages, provides and manages server side programming.

The web servers are

- 1) Apache Tomcat Webserver
 - provided by Apache SW foundation
 - It is fully written in Java.
 - Used for Servlets & JSPs.
 - The current version is Tomcat 9
- 2) IIS Webserver
 - Internet Information Services
 - for Windows default
 - Microsoft product
- 3) Light Speed Webserver
- 4) BEA Weblogic server
- 5) Google Web server
- 6) Netware etc.

Installation process

Apache Tomcat Web Server

Date
Page

- ① To install any Tomcat version, first we need to install JDK and must require support of JSR (Java S/w Development kit) and JRE (Java runtime environment).
- ② Go to tomcat.apache.org
- ③ Select Required Version to download
- ④ download Windows Service installer
- ⑤ ↴ Install the Tomcat
- ⑥ Accept license agreement, click on Agree
- ⑦ Choose components or features for Tomcat
- ⑧ Configure Tomcat by assigning Port number, Username and password.
- ⑨ Choose JVM Path location
c:\Program files\Java\jre (done automatically)
- ⑩ Choose installation location.
c:\prog files\Apache s/w Foundation\Tomcat 7.0

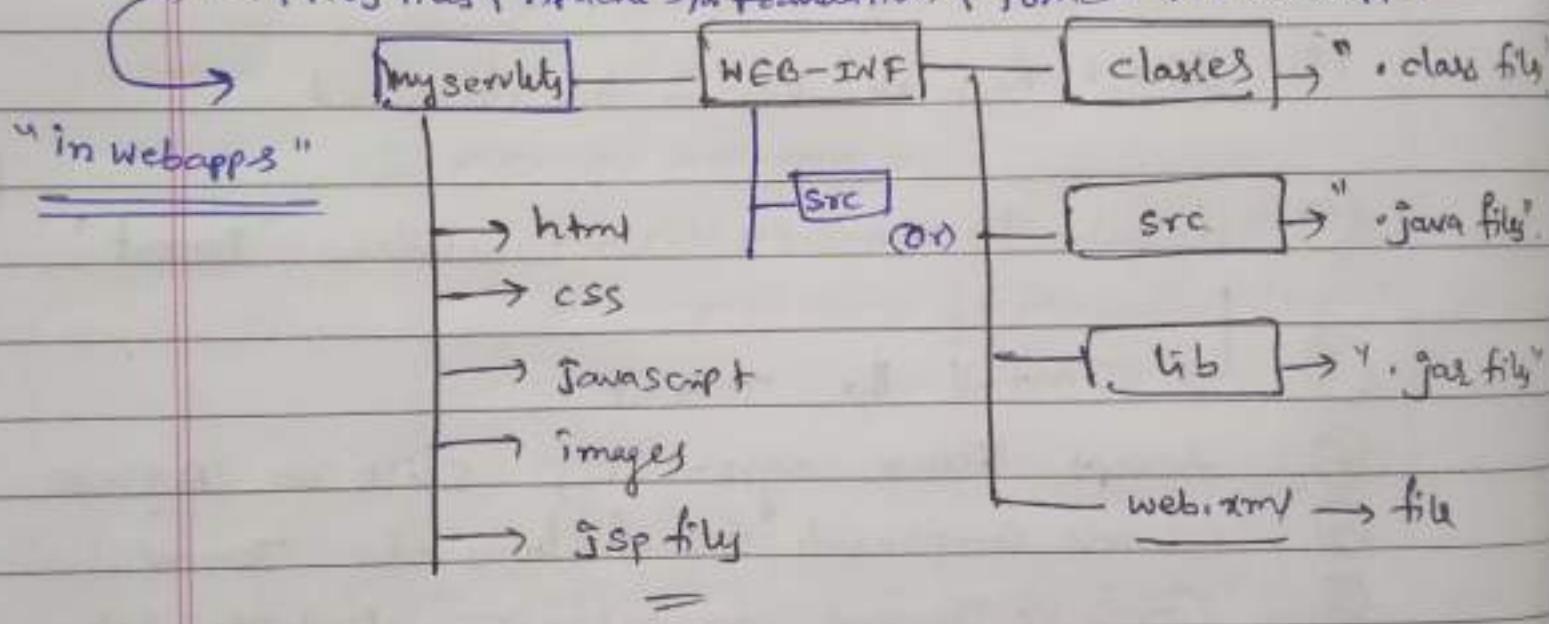
P.T.O.

To run any web application on Tomcat Server
 The following directory structure is used.

① Select a Drive for workspace

② Create a Directory in Tomcat i.e.

c:\Prog files\Apache S/W foundation\Tomcat 7.0\webapps



Path settings

① CATALINA_HOME

c:\program files\Apache S/W Foundation\Tomcat 7.0\

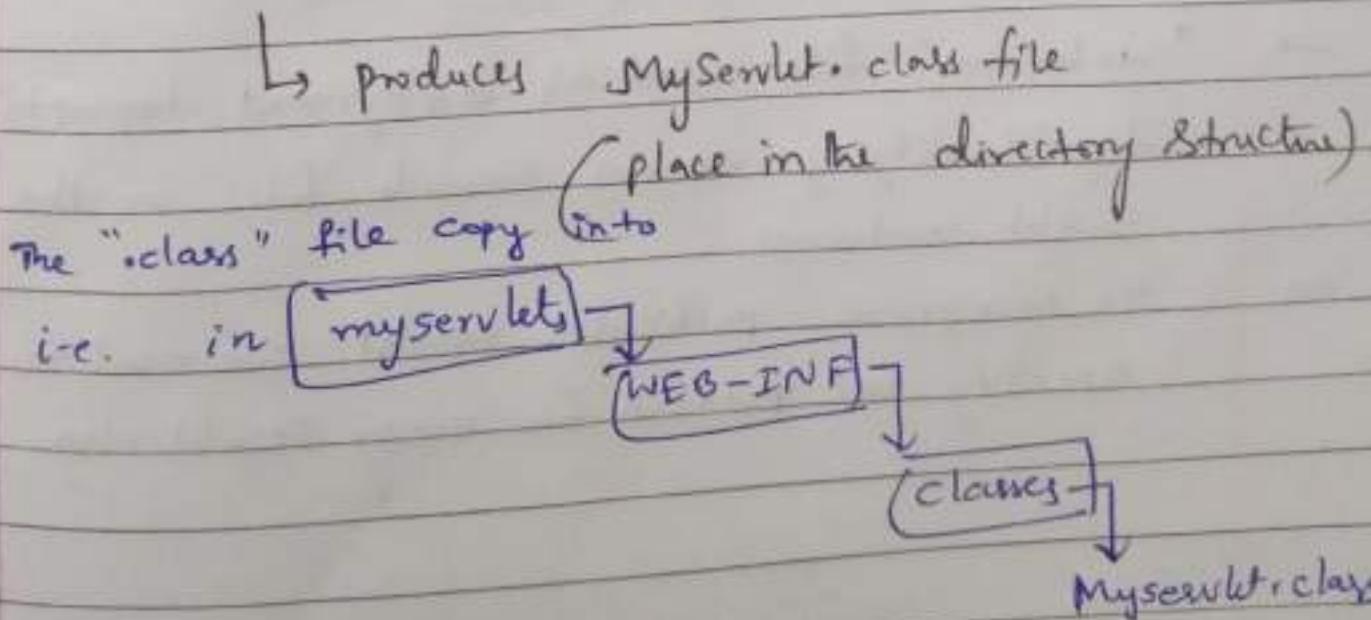
② classpath

c:\program files\Apache S/W Foundation\Tomcat 7.0\lib
 servlet-api.jar ; ↗

A Simple Servlet

```
import java.io.*;
import javax.servlet.*;
public class MyServlet extends GenericServlet
{
    public void service (ServletRequest req,
                         ServletResponse res)
        throws ServletException, IOException
    {
        PrintWriter pw = res.getWriter();
        pw.println (" welcome to Servlets ");
    }
}
```

Compile \Rightarrow javac MyServlet.java



place MyServlet.java is —

web.xml

```
<?xml version="1.0" ?>
<web-app>
  <servlet>
    <servlet-name> First </servlet-name>
    <servlet-class> MyServlet </servlet-class>
    </servlet>
    <servlet-mapping>
      <servlet-name> First </servlet-name>
      <url-pattern> /demo </url-pattern>
    </servlet-mapping>
  </web-app>
```

logical name
↳ class name
↳ url pattern.

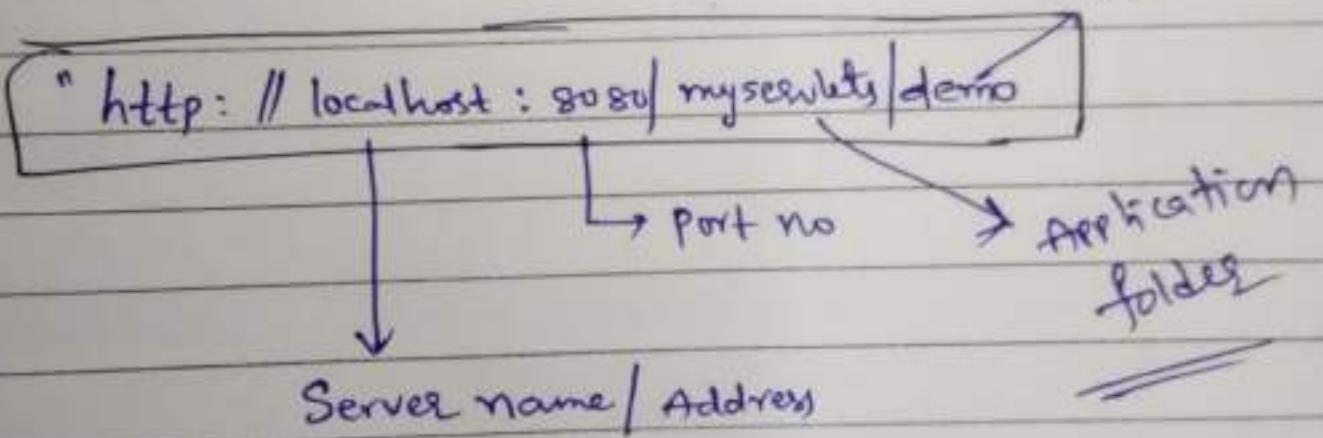
" Copy "web.xml" in "WEB-INF" folder

→ "web.xml" file used as deployment descriptor
for identifying various servlets files in the
Servlet container .

→ To recognize particular servlet we need to
specify url pattern for every servlet class .

To Execute a Servlet

- ① Create a directory structure under Tomcat for the application
- ② Write the Servlet source code
- ③ Compile the source code and place in classes of directory structure to ".class" file
- ④ Create a deployment descriptor "web.xml" file and place in [WEB-INF].
- ⑤ Start & Run Tomcat
- ⑥ Call the Servlet from web browser.



Servlet API

The **javax.servlet** and **javax.servlet.http** packages represent interfaces and classes for servlet api.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

1. javax.servlet Package

The javax.servlet package contains a number of interfaces and classes that establish the framework in which servlets operate.

The **interfaces** defined by the package are

Interface Name	Description
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.

1. Servlet Interface

All servlets must implement the **Servlet** interface. It declares the `init()`, `service()`, and `destroy()` methods that are called by the server during the life cycle of a servlet.

The methods defined are

Method Name	Description
<code>void init(ServletConfig sc) throws ServletException</code>	Called when the servlet is initialized. Initialization parameters for the servlet can be obtained from sc .
<code>void service(ServletRequest req, ServletResponse res) throws ServletException, IOException</code>	Called to process a request from a client. The request from the client can be read from req . The response to the client can be written to res .
<code>void destroy()</code>	Called when the servlet is unloaded.
<code>ServletConfig getServletConfig()</code>	Returns a ServletConfig object that contains any initialization parameters.
<code>String getServletInfo()</code>	Returns a string describing the servlet.

2. ServletConfig

The **ServletConfig** interface allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are

Method Name	Description
ServletContext getServletContext()	Returns the context for this servlet.
String getInitParameter(String param)	Returns the value of the initialization parameter named param .
Enumeration<String> getInitParameterNames()	Returns an enumeration of all initialization parameter names.
String getServletName()	Returns the name of the invoking servlet.

3. ServletContext

The **ServletContext** interface enables servlets to obtain information about their environment. Several of its methods are

Method Name	Description
Object getAttribute(String attr)	Returns the value of the server attributes named attr .
String getMimeType(String file)	Returns the MIME type of file .
String getRealPath(String vpath)	Returns the real path that corresponds to the virtual path vpath .
String getServerInfo()	Returns information about the server.
void log(String s)	Writes s to the servlet log.
void log(String s, Throwable e)	Writes s and the stack trace for e to the servlet log.
void setAttribute(String attr, Object val)	Sets the attribute specified by attr to the value passed in val .

4. ServletRequest

The **ServletRequest** interface enables a servlet to obtain information about a client request. Several of its methods are

Method Name	Description
Object getAttribute(String attr)	Returns the value of the attribute named attr .
Int getContentLength()	Returns the size of the request. The value -1 is returned if the size is unavailable.
String getContentType()	Returns the type of the request. A null value is returned if the type cannot be determined.
ServletInputStream getInputStream()	Returns a ServletInputStream that can be

<code>throws IOException</code>	used to read binary data from the request
<code>BufferedReader getReader() throws IOException</code>	Returns a buffered reader that can be used to read text from the request.
<code>String getParameter(String pname)</code>	Returns the value of the parameter named pname.
<code>Enumeration<String> getParameterNames()</code>	Returns an enumeration of the parameter names for this request.
<code>String[] getParameterValues(String name)</code>	Returns an array containing values associated with the parameter specified by name.
<code>String getProtocol()</code>	Returns a description of the protocol.
<code>String getRemoteAddr()</code>	Returns the string equivalent of the client IP address.
<code>String getRemoteHost()</code>	Returns the string equivalent of the client host name.
<code>String getScheme()</code>	Returns the transmission scheme of the URL used for the request (for example, "http", "ftp").
<code>String getServerName()</code>	Returns the name of the server.
<code>int getServerPort()</code>	Returns the port number.

5. ServletResponse

The `ServletResponse` interface enables a servlet to formulate a response for a client. Some of its methods are

Method Name	Description
<code>String getCharacterEncoding()</code>	Returns the character encoding for the response.
<code>ServletOutputStream getOutputStream() throws IOException</code>	Returns a <code>ServletOutputStream</code> that can be used to write binary data to the response.
<code>PrintWriter getWriter() throws IOException</code>	Returns a <code>PrintWriter</code> that can be used to write character data to the response.
<code>void setContentLength(int size)</code>	Sets the content length for the response to size.
<code>void setContentType(String type)</code>	Sets the content type for the response to type.

The **classes defined** by the package are

Class Name	Description
GenericServlet	Implements the Servlet and ServletConfig interfaces.
ServletInputStream	Provides an input stream for reading requests from a client.
ServletOutputStream	Provides an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable.

1. GenericServlet class

The **GenericServlet** class provides implementations of the basic life cycle methods for a servlet. **GenericServlet** implements the **Servlet** and **ServletConfig** interfaces.

2. ServletInputStream class

The **ServletInputStream** class extends **InputStream**. It is implemented by the servlet container and provides an input stream that a servlet developer can use to read the data from a client request.

int readLine(byte[] buffer, int offset, int size) throws IOException

Here, **buffer** is the array into which size bytes are placed starting at **offset**. The method returns the actual number of bytes read or **-1** if an end-of-stream condition is encountered.

3. ServletOutputStream class

The **ServletOutputStream** class extends **OutputStream**. It is implemented by the servlet container and provides an output stream that a servlet developer can use to write data to a client response. A default constructor is defined. It also defines the **print()** and **println()** methods, which output data to the stream.

4. ServletException

which indicates that a servlet problem has occurred.

5. UnavailableException class

which extends **ServletException**. It indicates that a servlet is unavailable.

2. javax.servlet.http Package

When working with HTTP, we will normally use the interfaces and classes in **javax.servlet.http**. As you will see, its functionality makes it easy to build servlets that work with HTTP requests and responses.

The **interfaces** defined by the package are

Interface Name	Description
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session.

1. HttpServletRequest Interface

The **HttpServletRequest** interface enables a servlet to obtain information about a client request. Several of its methods are

Method Name	Description
String getAuthType()	Returns authentication scheme.
Cookie[] getCookies()	Returns an array of the cookies in this request.
String getMethod()	Returns the HTTP method for this request.
String getPathInfo()	Returns any path information that is located after the servlet path and before a query string of the URL.
String getQueryString()	Returns any query string in the URL.
String getRemoteUser()	Returns the name of the user who issued this request.
String getRequestedSessionId()	Returns the ID of the session.
StringBuffer getRequestURL()	Returns the URL.
String getServletPath()	Returns that part of the URL that identifies the servlet.
HttpSession getSession()	Returns the session for this request. If a session does not exist, one is created and then returned.
HttpSession getSession(boolean new)	If new is true and no session exists, creates and returns a session for this request. Otherwise, returns the existing session for this request.
boolean isRequestedSessionIdValid()	Returns true if the requested session ID is valid in the current session context.

2. HttpServletResponse Interface

The `HttpServletResponse` interface enables a servlet to formulate an HTTP response to a client. The methods are

Method Name	Description
<code>void addCookie(Cookie cookie)</code>	Adds cookie to the HTTP response.
<code>boolean containsHeader(String field)</code>	Returns true if the HTTP response header contains a field named <code>field</code> .
<code>void sendError(int c) throws IOException</code>	Sends the error code <code>c</code> to the client.
<code>void sendError(int c, String s) throws IOException</code>	Sends the error code <code>c</code> and message <code>s</code> to the client.
<code>void sendRedirect(String url) throws IOException</code>	Redirects the client to <code>url</code> .
<code>void setStatus(int code)</code>	Sets the status code for this response to <code>code</code> .

3. HttpSession Interface

The `HttpSession` interface enables a servlet to read and write the state information that is associated with an HTTP session. Several of its methods are

Method Name	Description
<code>Object getAttribute(String attr)</code>	Returns the value associated with the name passed in <code>attr</code> . Returns null if <code>attr</code> is not found.
<code>Enumeration<String> getAttributeNames()</code>	Returns an enumeration of the attribute names associated with the session.
<code>long getCreationTime()</code>	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when this session was created.
<code>long getLastAccessedTime()</code>	Returns the time when the client last made a request for this session.
<code>String getId()</code>	Returns the session ID.
<code>void invalidate()</code>	Invalidates this session and removes it from the context.
<code>boolean isNew()</code>	Returns true if the server created the session and it has not yet been accessed by the client.
<code>void removeAttribute(String attr)</code>	Removes the attribute specified by <code>attr</code> from the session.
<code>void setAttribute(String attr, Object val)</code>	Associates the value passed in <code>val</code> with the attribute name passed in <code>attr</code> .

4. HttpSessionBindingListener Interface

The HttpSessionBindingListener interface is implemented by objects that need to be notified when they are bound to or unbound from an HTTP session. The methods that are invoked when an object is bound or unbound are

void valueBound(HttpSessionBindingEvent e)

void valueUnbound(HttpSessionBindingEvent e)

Here, e is the event object that describes the binding.

The classes defined by the package are

Class Name	Description
Cookie	Allows state information to be stored on a client machine.
HttpServlet	Provides methods to handle HTTP requests and responses.
HttpSessionEvent	Encapsulates a session-changed event.
HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a session value, or that a session attribute changed.

1. Cookie class

The Cookie class encapsulates a cookie. A cookie is stored on a client and contains state information. Cookies are valuable for tracking user activities.

The constructors for Cookie are

Cookie()

Cookie(String name, String value)

Here, the name and value of the cookie are supplied as arguments to the constructor.

The methods of the Cookie class are

Method Name	Description
String getComment()	Returns the comment.
String getDomain()	Returns the domain.
int getMaxAge()	Returns the maximum age (in seconds).
String getName()	Returns the name.
String getPath()	Returns the path.

<code>String getValue()</code>	Returns the value.
<code>int getVersion()</code>	Returns the version
<code>void setComment(String c)</code>	Sets the comment to c.
<code>void setDomain(String d)</code>	Sets the domain to d.
<code>void setMaxAge(int secs)</code>	Sets the maximum age of the cookie to secs. This is the number of seconds after which the cookie is deleted.
<code>void setPath(String p)</code>	Sets the path to p
<code>void setValue(String v)</code>	Sets the value to v.
<code>void setVersion(int v)</code>	Sets the version to v.

2. HttpServlet Class

The `HttpServlet` class extends `GenericServlet`. It is commonly used when developing servlets that receive and process HTTP requests. The methods defined by the `HttpServlet` class are

Method Name	Description
<code>void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException</code>	Handles an HTTP GET request.
<code>void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException</code>	Handles an HTTP POST request.
<code>void doPut(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException</code>	Handles an HTTP PUT request.
<code>void doDelete(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException</code>	Handles an HTTP DELETE request.
<code>void service(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException</code>	Called by the server when an HTTP request arrives for this servlet. The arguments provide access to the HTTP request and response, respectively.

3. HttpSessionEvent class

HttpSessionEvent encapsulates session events. It extends EventObject and is generated when a change occurs to the session.

The constructors are

HttpSessionEvent(HttpSession session)

Here, session is the source of the event.

The method defined is

HttpSession getSession() - returns the session in which event occurred.

4. HttpSessionBindingEvent class

The HttpSessionBindingEvent class extends HttpSessionEvent. It is generated when a listener is bound or unbound in an HttpSession object. It is also generated when an attribute is bound or unbound.

HttpSessionBindingEvent(HttpSession session, String name)

HttpSessionBindingEvent(HttpSession session, String name, Object val)

Here, session is the source of the event, and name is the name associated with the object that is being bound or unbound. If an attribute is being bound or unbound, its value is passed in val.

Method Name	Description
String getName()	obtains the name that is being bound or unbound
HttpSession getSession()	obtains the session to which the listener is being bound or unbound:
Object getValue()	obtains the value of the attribute that is being bound or unbound.

Reading Servlet Parameters

<!-- Reading Servlet parameters or accessing form data -->

```
<html>
<body>
<center>
<form name="f1" method="post" action="http://localhost:8080/mydemos/read">
<table>
<tr>
<td><B>User Name:</td>
<td><input type="text" name="un" size="25" value=""></td>
</tr>
<tr>
<td><B>Password:</td>
<td><input type="text" name="pw" size="25" value=""></td>
</tr>
</table>
<input type="submit" value="Login">
</body>
</html>
```

<!—Deployment Descriptor web.xml-->

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app>
    <servlet>
        <servlet-name>Third</servlet-name>
        <servlet-class>ReadServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Third</servlet-name>
        <url-pattern>/read</url-pattern>
    </servlet-mapping>
</web-app>
```

```
//Reading Servlet Parameters or Accessing form data
import java.io.*;
import java.util.*;
import javax.servlet.*;
public class ReadServlet extends GenericServlet
{
    public void service(ServletRequest req , ServletResponse res)
        throws ServletException , IOException
    {
        PrintWriter pw = res.getWriter();

        Enumeration e = req.getParameterNames(); //gets all parameter names

        pw.println("The user details are");

        while(e.hasMoreElements())
        {
            String pname = (String)e.nextElement();
            pw.print(pname + " = ");
            String pvalue = req.getParameter(pname);
            pw.println(pvalue);
        }

        String name = req.getParameter("un"); //gets a value of specified parameter
        String pwd= req.getParameter("pw");

        if(name.equals("ramesh") && pwd.equals("vce"))
            pw.println("Welcome to " +name);
        else
            pw.println("Invalid User Details");

        pw.close();
    }
}
```

Reading Initialization Parameters

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class InitServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {

        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();

        ServletConfig con=getServletConfig();
        Enumeration<String> e=con.getInitParameterNames();

        String str="";
        while(e.hasMoreElements())
        {
            str=e.nextElement();
            pw.println("<br>Name: "+str);
            pw.println(" value: "+con.getInitParameter(str));
        }

        pw.println("The Drivername is " +con.getInitParameter("driver"));
        pw.close();
    }
}

<!—Deployment Descriptor web.xml-->

<web-app>
    <servlet>
        <servlet-name>InitPar</servlet-name>
        <servlet-class>InitServlet</servlet-class>

        <init-param>
            <param-name>driver</param-name>
            <param-value>jdbc:mysql://localhost:3306/employee</param-value>
        </init-param>
        <init-param>
```

```

<param-name>username</param-name>
<param-value>ramesh</param-value>
</init-param>

<init-param>
<param-name>password</param-name>
<param-value>root</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>InitPar</servlet-name>
<url-pattern>/init</url-pattern>
</servlet-mapping>

</web-app>

```

Handling HTTP Requests and Responses

1. Handling HTTP GET Requests

/ Handling HTTP Requests and Responses The HttpServlet class provides specialized methods that handle the various types of HTTP requests. A servlet developer typically overrides one of these methods. These methods are doDelete(), doGet(), doHead(), doOptions(), doPost(), doPut(), and doTrace().
the GET and POST requests are commonly used when handling form input. */*

```

<!-- Handling HTTP GET requests -->

<html>
<body>
<center>
<form name="f1" method="get" action="http://localhost:8080/mydemos/getdata">
<table>
<tr>
<td><B>Student Name:</td>
<td><input type="text" name="sname" size="25" value=""></td>
</tr>
<tr>
<td><B>Branch:</td>
<td> <select name="branch" size="1">
<option value="CSE">Computer Science Engineering</option>
<option value="ECE">Electronics Communication Engg</option>
<option value="EEE">Electrical Engineering</option>

```

```

        </select>
    </td>
</tr>
</table>
<input type="submit" value="Register">
</body>
</html>

//Java Servlet Class
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HttpGet extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res)
            throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        String name=req.getParameter("sname");
        String course = req.getParameter("branch");
        pw.println("<b>The Studen </b>" +name);
        pw.println("<B>selected Course is: </b>" +course);
        pw.close();
    }
}

```

```

<!--Deployment Descriptor web.xml-->

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

    <servlet>
        <servlet-name>Fourth</servlet-name>
        <servlet-class>HttpGet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Fourth</servlet-name>
        <url-pattern>/getdata</url-pattern>
    </servlet-mapping>

</web-app>

```

2.Hadling HTTP POST Requests

```
<html>
<body>
<center>
<form name="f1" method="post" action="http://localhost:8080/mydemos/postdata">
<table>
<tr>
<td><B>Student Name:</td>
<td><input type="text" name="sname" size="25" value=""></td>
</tr>
<tr>
<td><B>Password:</td>
<td> <select name="branch" size="1">
            <option value="CSE">Computer Science Engineering</option>
            <option value="ECE">Electronics Communication Engg</option>
            <option value="EEE">Electrical Engineering</option>
        </select>
</td>
</tr>
</table>
<input type="submit" value="Register">
</body>
</html>

//Java Servlet class
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HttpPost extends HttpServlet
{
    public void doPost(HttpServletRequest req,HttpServletResponse res)
                throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        String name=req.getParameter("sname");
        String course = req.getParameter("branch");
        pw.println("<b>The Studen </b>" +name);
        pw.println("<B>selected Course is: </b>" +course);

        pw.close();
    }
}
```

```
<!--Deployment Descriptor web.xml-->
```

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

    <servlet>
        <servlet-name>Fifth</servlet-name>
        <servlet-class>HttpPost</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Fifth</servlet-name>
        <url-pattern>/postdata</url-pattern>
    </servlet-mapping>
</web-app>
```

Accessing a Database from Servlet

```
//Accessing EMP table data - Database from servlet
```

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class EmpAccess extends HttpServlet
{
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException
    {
        PrintWriter pw = res.getWriter();
        res.setContentType("text/html");
        //String tableName = req.getParameter("table");
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/employee","root","");
            Statement st=con.createStatement();
            ResultSet rs = st.executeQuery("select * from emp");
            pw.println("<table border=2>");
            while(rs.next())
            {
                pw.println("<tr><td>" + rs.getInt(1) + "</td>");
                pw.println("<td>" + rs.getString(2) + "</td>");
                pw.println("<td>" + rs.getString(3) + "</td>");
                pw.println("<td>" + rs.getInt(4) + "</td></tr>");
            }
            pw.println("</table>");
            pw.close();
        }
```

```
        con.close();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}
```

//Inserting into Database using Servlet

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class InsertData extends HttpServlet
{
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException
    {
        PrintWriter pw = res.getWriter();
        res.setContentType("text/html");
        /* int eid = Integer.parseInt(req.getParameter("id"));
           String ename = req.getParameter("name");
           String edept = req.getParameter("dept");
           int esal = Integer.parseInt(req.getParameter("sal"));
           pw.println(eid);
           pw.println(ename);
           pw.println(edept);
           pw.println(esal); */

        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/employee","root","");
            Statement st=con.createStatement();
            st.executeUpdate("INSERT INTO emp(id,name,dept,salary)    VALUES
(445,'ramgopal','designer',34568)");

            pw.close();
            con.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

//Accessing date based on empid

```
<html>
<head>
<title> Employee Details </title>
</head>
<body>
<center>
<form name="f1" method="get" action="http://localhost:8080/mydemos/gets">
Enter Employee id:<input type="text" name="t1">
<input type="submit" value="find employee details">
</form>
</center>
</body>
</html>

import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AccessDemo extends HttpServlet
{
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        PrintWriter pw = res.getWriter();
        res.setContentType("text/html");
        int empid=Integer.parseInt(req.getParameter("t1"));

        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee","root","");
PreparedStatement st=con.prepareStatement("select * from emp where id=?");
            st.setInt(1,empid);
            ResultSet rs=st.executeQuery();

            pw.println("<table border=2>");
            while(rs.next())
            {
                pw.println("<tr><td>" + rs.getInt(1) + "</td>");
                pw.println("<td>" + rs.getString(2) + "</td>");
                pw.println("<td>" + rs.getString(3) + "</td>");
                pw.println("<td>" + rs.getInt(4) + "</td></tr>");
            }
            pw.println("</table>");
            pw.close();
            con.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Handling Cookies in Java

A cookie is a small piece of information that is persisted between the multiple client requests. Used to maintain the state information at client side.

A cookie is stored on a client and contains state information. Cookies are valuable for tracking user activities.

By default, each request is considered as a new request.

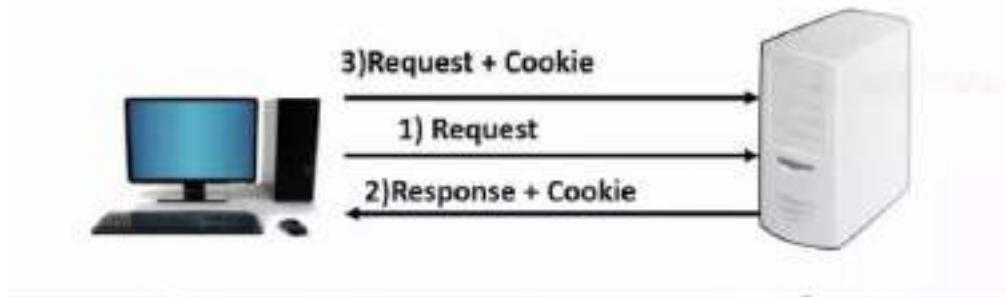
Example: When a user visits an online store. A cookie can save the user's name, address, and other information. The user does not need to enter this data each time visits the store.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

A Cookie functions as

1. For the first response the server (servlet) add cookie, So the cookie is stored in the cache of the browser.
2. After that if another request is sent by the user, the cookie is added with request by default by the browser. Thus, the server recognizes the user as the old or same user.

Cookie



The two types of cookies in servlets are

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 sessions. It is not removed each time when user closes the browser. It is removed only if user logout or sign-out.

Disadvantage of Cookies

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

The Cookie process is

1. The servlet creates a Cookie object based on the client request using **Cookie** class.
2. A servlet can write a cookie to a user's machine via the **addCookie()** method of the **HttpServletResponse** interface. The data for that cookie is then included in the header of the HTTP response that is sent to the browser.

The names and values of cookies are stored on the user's machine. Some of the information that is saved for each cookie includes the following:

- The name of the cookie
- The value of the cookie
- The expiration date of the cookie
- The domain and path of the cookie

The expiration date determines when this cookie is deleted from the user's machine. If an expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends.

Cookie class

`javax.servlet.http.Cookie` is used to create and manage cookies.

The Constructors are

<code>Cookie()</code>	Constructs a cookie.
<code>Cookie(String name, String value)</code>	Constructs a cookie with a specified name and value.

The Methods of Cookie class are

Method	Description
<code>String getName()</code>	Returns the name.
<code>int getMaxAge()</code>	Returns the maximum age (in seconds).
<code>String getPath()</code>	Returns the path.
<code>String getValue()</code>	Returns the value.
<code>int getVersion()</code>	Returns the version.
<code>void setMaxAge(int secs)</code>	Sets the maximum age of the cookie to <i>secs</i> . This is the number of seconds after which the cookie is deleted.
<code>void setPath(String p)</code>	Sets the path to <i>p</i> .
<code>void setValue(String v)</code>	Sets the value to <i>v</i> .
<code>void setVersion(int v)</code>	Sets the version to <i>v</i> .

//Example Program to Handle Cookie

<!-- Creates Cookie with user name -->

```
<html>
<body>
<center>
```

```
<form name="f1" method="post"
action="http://localhost:8080/mydemos/addck">
<table>
<tr>
<td><B>User Name:</td>
<td><input type="text" name="un" size="25" value=""></td>
</tr>
<tr>
<td><B>Mobile No:</td>
<td><input type="text" name="no" size="25" value=""></td>
</tr>
</table>
<input type="submit" value="Submit">
</body>
</html>
```

//Create and Add Cookie in Response

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AddCookie extends HttpServlet
{
    public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException

    {
        String name = req.getParameter("un");
        Cookie ck = new Cookie("FirstCookie", name);
        ck.setMaxAge(24*60*60);
        res.addCookie(ck);
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        pw.println("<B>Cookie is Created and sent to Client Browser");
        pw.println(name);
        pw.println("Welcome to " +name);
        pw.close();
    }

}
```

```
//Read Cookie information from Browser
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ReadCookie extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws ServletException, IOException
    {
        Cookie[] ck = req.getCookies();
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        pw.println("<B>");
        for(int i = 0; i < ck.length; i++)
        {
            String name = ck[i].getName();
            String value = ck[i].getValue();
            pw.println(name + ":" + value);
        }
        pw.close();
    }
}
```

Session Tracking

- ✓ Session means a particular interval of time.
- ✓ Session Tracking is a way to maintain state (data) of a user. It is also known as session management in servlet.
- ✓ The session is stored in and managed by server.
- ✓ Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of a user to recognize to particular user.
- ✓ HTTP is a stateless protocol. Each request is independent of the previous one. In some applications, it is necessary to save state information so that information can be collected from several interactions between a browser and a server.
- ✓ Sessions provide such a mechanism for session management.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DateServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req,
                      HttpServletResponse res)
                      throws ServletException, IOException
    {
        HttpSession hs = req.getSession(true);
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        pw.print("<B>");
        // Display date/time of last access.
        Date date = (Date)hs.getAttribute("date");

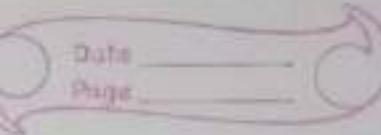
        if(date != null)
        {
            pw.print("Last access: " + date + "<br>");
        }
    }
}
```

```
// Display current date/time.  
date = new Date();  
hs.setAttribute("date", date);  
pw.println("Current date: " + date);  
}  
}
```

//First Request only Current Date and Time
//Next request from same browser is displays
Previous Session Time
Current Session time.

UNIT - IV

"Java Server Pages".



- ✓ Java Server Pages (JSP) is a Server-side programming technology that enables the creation of dynamic, platform-independent methods for building Web-based applications.
- ✓ JSP has access to complete Java APIs, JDBC etc.

Introduction to JSP

- JSP is a Server side programming technology for building dynamic web pages.
- It provides more functionality than Servlets.
- A JSP file consists of HTML tags & JSP tags.
- A JSP file is saved with file extension ".jsp".
- Helps developers insert Java code in HTML pages by using special JSP tags.

The Problems With Servlets.

- ① A Single Servlet class has to perform tasks such as
 - ✓ Accepting the request
 - ✓ Processing of request
 - ✓ Handling the logic
 - ✓ Generation of (HTML) response.
- ② For developing web based application, The developer must have knowledge of Java as well as HTML code.
- ③ If a small change in the look and feel of Web based application, the whole application is recompiled, configured and executed.
- ④ Servlets do not support the use of web based tools for application development . with this applying complete styles & look & feel is more complex, time consuming & prone to errors.

pw.println(" Welcome ");
(Macromedia flash | dream viewer)

All these are problems associated with Servlets because of single servlet to handle all tasks.

These problems are resolved using JSP Technology.

The Anatomy of JSP Page (Structure)

JSP Page is a Simple web page which contains

① Template Text - It can be Script code

Such as HTML, CSS, XML

or a Simple text

② JSP elements - Responsible for dynamic Content. The various

JSP elements can be action tags, custom tags, directives, JSTL library elements etc.

(JSTL - Java Server Pages Standard Tag Library).

```
<%@ page language = "java" contentType = "text/html" %>
```

```
<html>
```

```
  <head>
```

```
    <title> First Jsp Page </title>
```

```
  </head>
```

```
  <body bgcolor = "green" >
```

```
    <H1> welcome to the Day </H1>
```

```
    <H1> <% = new Date().toString() %> </H1>
```

```
  </body>
```

```
</html>
```

Template Text

JSP elements

After processing JSP request, The template text and JSP elements are merged together, Sent to browser as response.

"JSP Processing"

- To process JSP pages , a web server needs a JSP engine called JSP Container.
- The JSP Container is responsible for handling JSP pages.
- The JSP Container works with the webserver to provide runtime environment and the services needed by JSP.

The JSP processing is

- ① The browser sends the HTTP requests to the Web Server.
- ② The Web Server recognizes the HTTP request is for a JSP page and forwards the request to JSP container . (By viewing .jsp file)
Eg: ~~do~~ "first.jsp"
- ③ On receiving the request , the JSP container searches & reads the desired page (JSP file) .

The JSP page is converted into corresponding Servlet .

The JSP page is combination of JSP elements and template text .

The template text is converted into corresponding println statement.

Eg: `<html>` } `pw.println("<html>");`
`<body>` } \Rightarrow `pw.println("<body>");`
=

Every JSP element is converted into corresponding java code.

This phase is called "Translation Phase"

The output of 'Translation Phase' is a Servlet.

Eg: `first.jsp` \longrightarrow `firstServlet.java`.

④ The Jsp Container Compiles the Servlet and Produces the ".class" file [Eg: `firstServlet.class`], forwards the original request to Servlet engine.

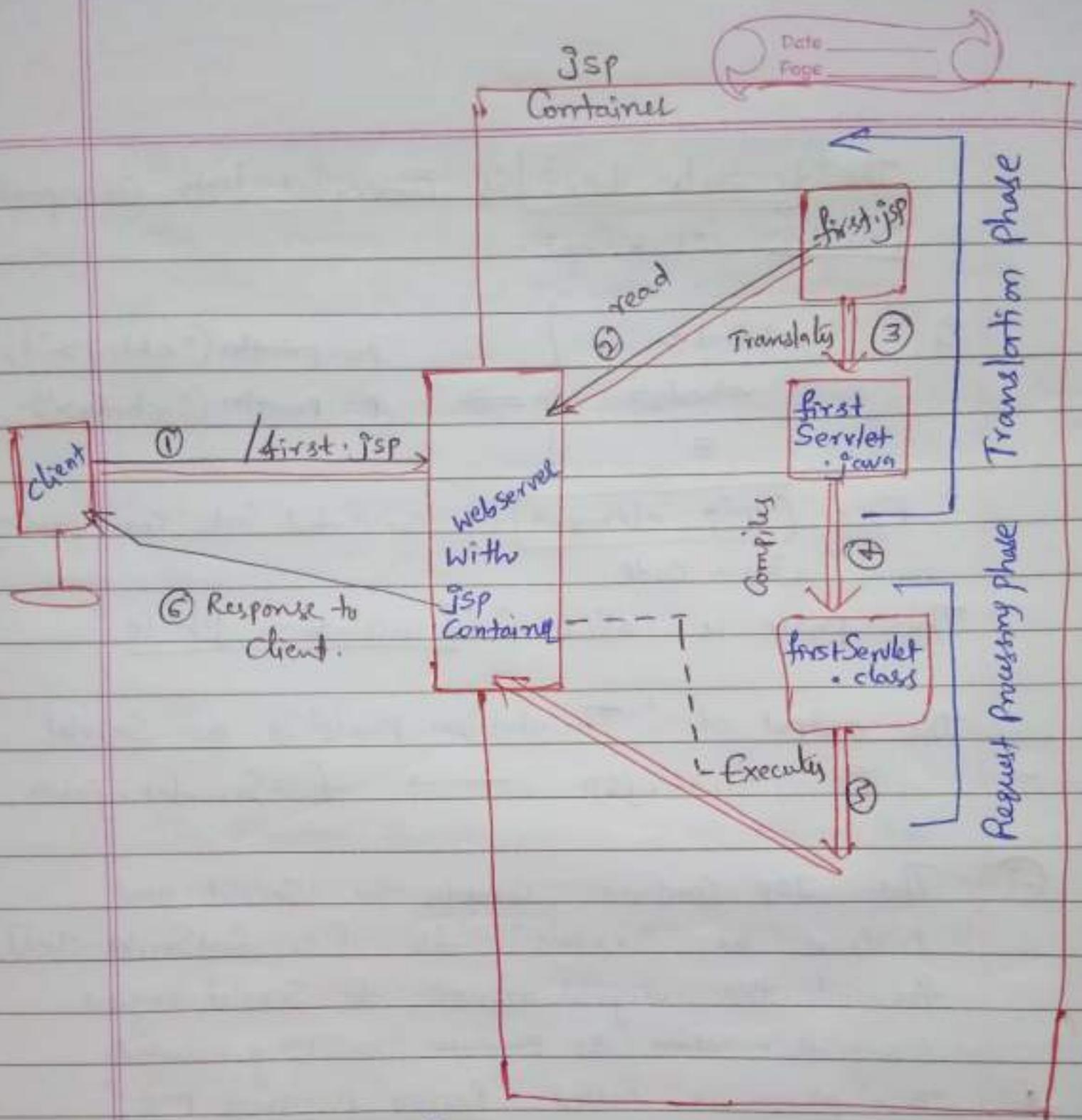
Using this class, the Response can be generated.

This phase is called "Request Processing Page".

⑤ The Servlet Container loads the Servlet and executes it, produces the Output in HTML format.

⑥ The Container give response to webserver.

⑦ The webserver forwards the HTTP response to browser.



Only for the first request, the response is slow. for every next request, if it is to the same JSP page, then the Container directly executes and gets response. Thus improves the performance.

JSP Application Design with MVC.

MVC stands for Model-View-Controller Architecture, it is a design pattern for developing web applications.

The basic idea in MVC model is to separate design logic into three parts - Modelling, viewing and controlling.

A server application is classified into 3 parts.

① Model - (Business Logic)

The logic applied for manipulation of application data. i.e. It can have business logic. It is responsible for managing data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself. (Java Beans, EJB)

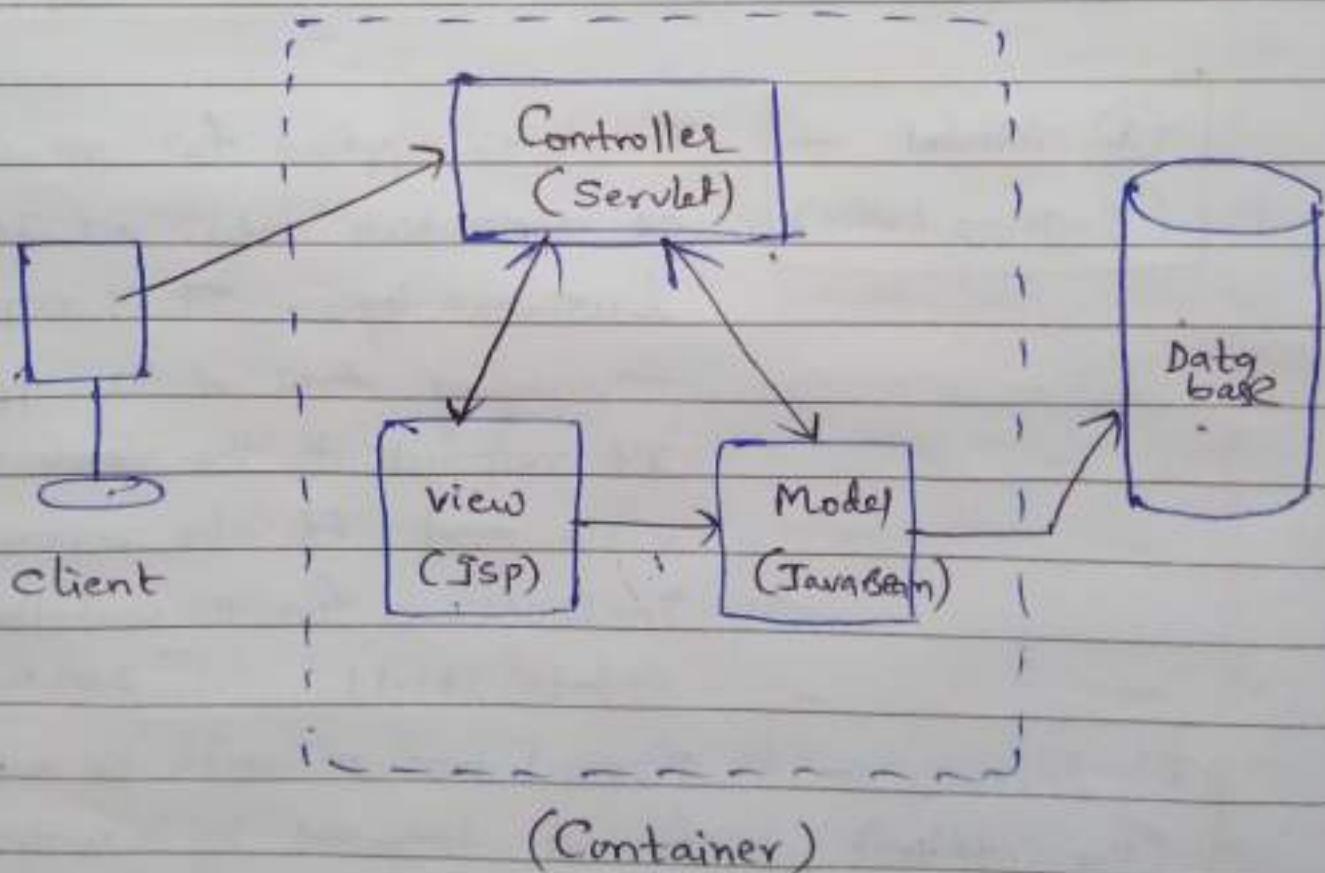
② View - (Presentation logic)

Presentation of data in a particular format, triggered by a controller's decision to present the data. It is the code written for look and feel of the webpage. (HTML, JSP, PHP) Eg: Background color, font size, label etc.

③ Controller —
(Request Processing)
logic

Responsible for responding
to the user input and
perform interactions on the
data model objects.

The controller receives the input, validates
and then performs business operations
that modifies state of data model.
(Servlet).



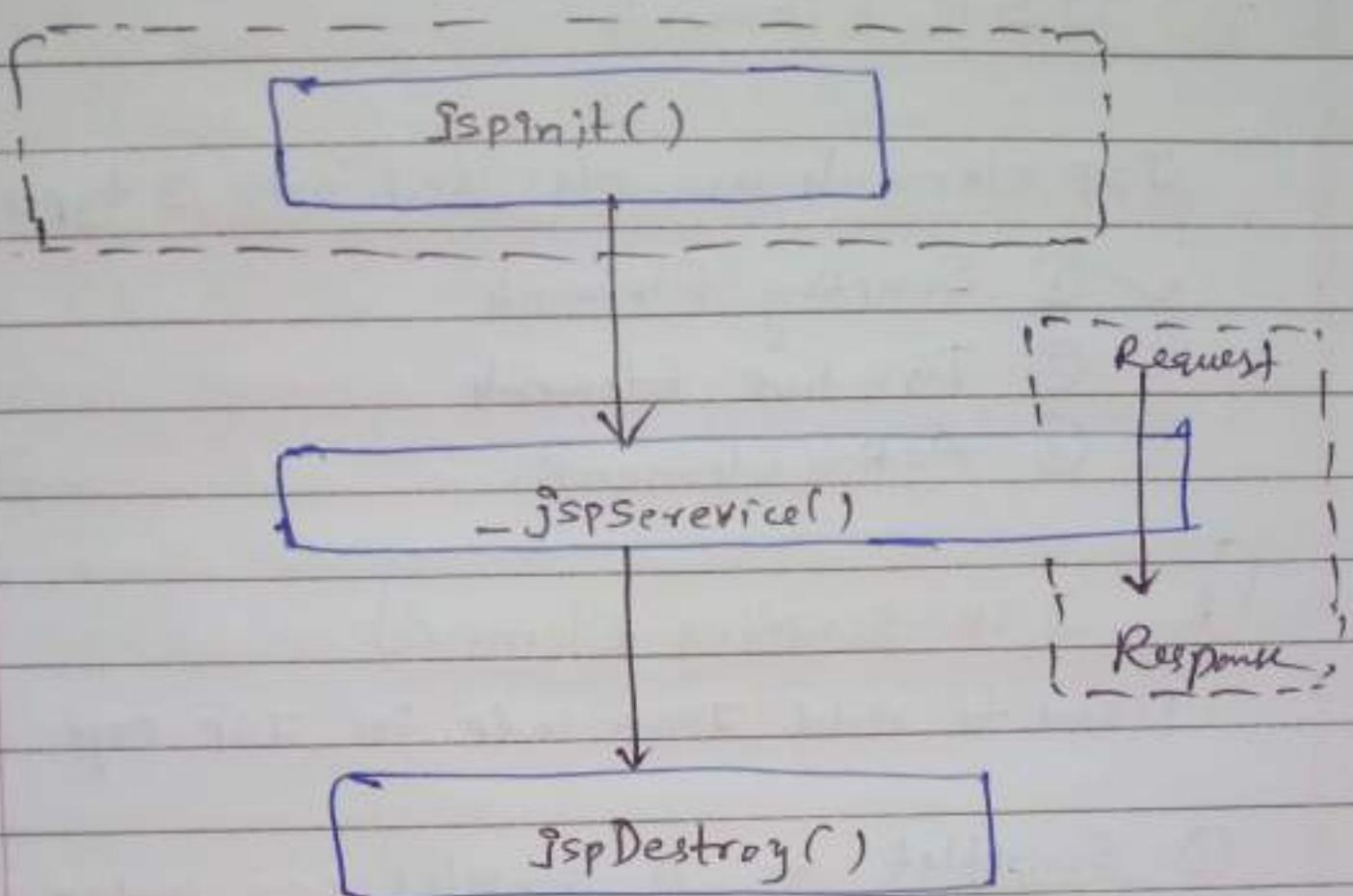
The Various Advantages of MVC Architecture

- ① Faster Web Application Development
- ② Ideal for developing large size web application
- ③ MVC model returns the data without the need of formatting.
- ④ Allows to use web development tools like Macromedia Flash or Dream Viewer.
- ⑤ The modifications never effect the entire model.
- ⑥ ~~Has~~ clear separation between business, presentation & request processing logic.

JSP - Life cycle.

The life cycle of Jsp is in 4 phases.

- ① Compilation - If the browser request a Jsp page, first checks is it required to compile | is it already compiled. If required
 - Parsing JSP page
 - Translate JSP into Servlet
 - Compiling the Servlet -
- ② JSP Initialization - The container invokes the `jspInit()` before servicing any requests. We can initialize database connections, open files, create tables etc.
- ③ JSP Execution - The container invokes `- jspService()` method in the JSP. It takes `HttpServletRequest` & `HttpServletResponse` as parameters. It is responsible for generating the response.
- ④ JSP Cleanup - Before a JSP page is unloaded, this method is called to perform any cleanup operation such as closing files or releasing database connections.



JSP Elements

Jsp elements are classified into 3 types

- ✓ ① Scripting Elements
- ② Directive Elements
- ③ Action Elements.

1. JSP Scripting Elements

Used to Add Java code in JSP page

① Scriptlet — A scriptlet can contain any number of Java language statements, variables or method declarations or expressions.

The Syntax of Scriptlet is

<% Code fragment %>

A JSP page can have any no. of scriptlet tags.

<html>
<body>

Eg: <% out.println ("Welcome to Jsp"); %>

<% out.println (" Nice day "); %>

<h1>

<% out.println ("Welcome to World"); %>

</h1>

<body>

</html>

② JSP Declarations — Declares one or more variables or methods that we can use in Java code. The variable must be declared before using them. The Syntax of declaration is

<%! declaration; [declaration;] # ... %>

It is called declaration tag, used to declare static members, instance variables or methods.

Eg.

<html>

<body>

<h3> Variables & Method Declaration </h3>

<%! String msg = "Hello World"; %>

<%! int a = 10; int b = 20; double c = 3.5; %>

<%! ^{public} double mul(double a, double b)

{

return (a * b);

%> }

<% out.println("The msg is " + msg); %>

<% out.println("The result is " + (a + b + c)); %>

<% out.println("The a value is " + a); %>

<% out.println("The function call " + mul(5.5, 3.5)); %>

</body>

</html>

Eg. <%! int f=0; %>
 <%! int a,b,c; %>
 <%! Box bi=new Box(10,20,30); %>

③ JSP Expression — This element contains a scripting language expression that is evaluated, converted to a string, inserted where the expression appears in the Jsp file.

The expression element contains any expression that is valid according to the java language specification.

NOTE : No semicolon to end an expression.

The Syntax of JSP expression is

<%= expression %>

Eg. <!-- Expression element -->
 <html>
 <body>
 <%= (200*8) %>
 <%= (new java.util.Date()) %>
 </body>
 </html>



④ JSP Comments - Marks the text or statements that the JSP container should ignore.

The syntax is

<%-- Jsp comments here --%>

Eg: <%-- The method to find volume --%>

Note: - HTML comments are also allowed.

<!-- The method to find volume -->

// Program to demonstrate factorial of a no.

```
<%@ page language = "java" contentType = "text/html" %>
<html>
    <body style = "color : red" >
        <%. out.println ("The factorial of 10 is"); %>
        <% int fa = 1; %>
        <%
            for (int i = 1; i <= 10; i++)
            {
                fa = fa * i;
            }
            out.println (" → " + fa);
        <%>
    </body>
</html>
```

2. Action Elements.

Date
Page

- JSP Action elements are useful to perform a specific task in JSP page.
- The actions use constructs in XML syntax to control the behaviour of servlet container.
- Actions used to dynamically insert a file, reuse Java bean components, forward user to another page etc.

"Action elements are predefined functions".

The Syntax is

<jsp:action-name attribute = "value" />

① <jsp:include>

- Used to include another JSP file or HTML file in to JSP page.
- Inserts a file at the time the JSP page is translated into a servlet, this action inserts the file at the time of page is requested.

- The Syntax is

<jsp:include page = "url" flush = "true" />

clears buffer
before loads.

Eg : <html>
 <body>

<h1> finding factorial of a number </h1>
<jsp:include page = "fact.jsp" flush = "true"/>
</body>
</html>

" It provides better reusability of the code ".

② <jsp:param>

- It is used to pass parameters to another file.
- It can be used with either <jsp:include> or <jsp:forward>.
- The Syntax is

<jsp:param name = "par-name" value = "" />

Eg : <jsp:param name = "n" value = "10" />
<jsp:param name = "city" value = "hyd" />

Eg : To find the reverse of a number.

<%@page language = "java" contentType = "text/html" %>
<html>
<body>
<jsp:include page = "reverse.jsp">
<jsp:param name = "n1" value = "123" />
</body> </jsp:include>

reverse.jsp

```
<%@ page language="java" contentType="text/html" %>
<html>
    <body>
        <% int n = Integer.parseInt(request.getParameter("n"));
           int s=0, r;
           while (n > 0)
           {
               r = n % 10 ;
               s = s * 10 + r;
               n = n / 10;
           }
           out.println ("The reverse num is " + s);
       </body>
   </html>
```

Execution : <http://localhost:8080/myjssps/param.html>.

③. <jsp:forward>

- Used to forward the request to a new page.
- Terminates the action of the current page and forwards the request to another resource such as html page, Jsp Page or Java servlet.
- The Syntax is
`<jsp:forward page = "URL of source" />`

Eg:

```

<html>
  <body>
    } → forward JSP
    <jsp:forward page = "reverse.jsp" />
    <jsp:param name = "n1" value = "123" />
    </jsp:forward>
  </body>
</html>

```

④. <jsp:plugin>

- Used to insert Java components in a Jsp page.
- Used to include Bean or Applet into a Jsp page.
- <param> element can also be used to send parameters to the applet or bean.
- The Syntax is

```

<jsp:plugin type = "bean/applet" code = "classfile"
             align = "left/middle/right" width = "pixels"
             height = "pixels" />

```

Eg:

If a specified plugin is not available,
then to display alternate text or error message
`<jsp: fallback>` is used.

The Syntax is

```
<jsp: fallback>
    // Error Msg
<| jsp: fallback>
<html>
    <body>
        <jsp: plugin type = "applet" code = "MyApplet.class"
            width = "300" height = "200" >
            <jsp: fallback>
                unable to load plugin
            <| jsp: plugin>
        <MyApplet.class>
            import java.awt.*;
            import java.applet.*;
            public class MyApplet extends Applet
            {
                public void paint (Graphics g)
                {
                    g.setBackground (Color.blue);
                    g.drawString ("It is a Applet", 10, 50);
                    g.drawRect (50, 100, 800, 40);
                }
            }
```

* Execute in
"internet explorer"

<jsp:useBean>

- It is used to find or instantiate a Java bean
- It searches for an existing object utilizing the 'id' and 'scope' variables. If the object is not found, it tries to create the specified object.
- The syntax is

```
<jsp:useBean id="name" class="package.class"/>
```

- Once a Bean class is loaded, "jsp:setProperty" and "jsp:getProperty" actions used to set or retrieve Bean properties.

<jsp:setProperty>

- Used to set the property of a bean.
- The Bean must have been previously defined before this action.
- The syntax is

```
<jsp:setProperty name="Bean name"  
property="PropertyName" value="anyval"/>
```

<jsp:getProperty>

- Used to retrieve the value of a given bean property and converts it to a String, finally insert it into output.
- The Syntax is

```
<jsp:getProperty name="Bean name" property="Prop name"/>
```

Eg: ③. // demonstrates Student data.

```
package Student;
public class StudData           StudData.java
{
    String name;
    public StudData()
    {
        name = "Vivek";
    }
    public void setName( String n )
    {
        name = n;
    }
    public String getName()
    {
        return name;
    }
}
```

To access in Jsp

```
<html>                                bean.jsp.
<body>
<jsp:useBean id="s1" class="student.StudData"/>
<% out.println( s1.getName()); %>
<% s1.setName("Ranush"); %>
<% out.println( s1.getName()); %>
</body>
</html>
```

② // Demonstrate Employee data.

```
Package emp;  
public class Employee  
{ int salary;  
    public Employee()  
    { salary = 25000;  
    }  
    public void setSalary(int x)  
    { salary = x;  
    }  
    public int getSalary()  
    { return(salary);  
    }
```

Employee.java

To access in JSP page

```
<html>  
  <body>  
    <jsp:useBean id="e1" class="emp.Employee" />  
    <jsp:getProperty name="e1" property="salary" />  
    <jsp:getProperty name="e1" property="salary"  
                     value="45000" />  
    <jsp:getProperty name="e1" property="salary" />  
  </jsp:useBean>  
  </body>  
</html>
```

<jsp:text>

- It is used to write template text in JSP pages and documents.
- The body of the template can not contain other elements. (only text & expressions).
- The Syntax is
`<jsp:text> template text </jsp:text>`

Implicit JSP objects

Date _____
Page _____

- During JSP translation phase, a web container creates built-in objects, called implicit objects.
- These objects are Java objects the JSP container makes available to the developer in each page and the developer can call them directly without explicitly being declared.
- Implicit objects are also called Predefined variables.
- The various implicit JSP objects are

1. request - , HttpServletRequest object associated with the request.

/ It provides methods for accessing the client request.

Eg: / getContentLength()

getServerName()

getParameter()

getParameterNames() etc.

<% String x = request.getParameter("user"); %>

2. response - , It is HttpServletResponse object associated with response to client

/ It provides the methods for adding cookies, session or response, setting header etc.

- / addCookie(), addHeader(),
- / setContentType(), getContentType() etc
- / sendRedirect().

Eg:

```
<%@ out.println ("welcome");  
response.setContentType ("text/html");  
response.setContentType ("text/html");  
response.sendRedirect ("Welcome.html");  
%> .
```

3. out - , This is the PrintWriter object used to send output to the client
- / It provides methods related to I/O.
 - / clear(), newline(), print(), println()
 - / The output object is created by JspWriter.
- ```
<%@ out.println ("Hello world");
```

4. page - , It is similar to "this" keyword in Java

- / It refers to the current JSP page.
- / It represents Object class.

- ⑤ exception - It is an object of Throwable.  
- This object is for handling error pages and contain information about runtime error.

<!-- exception -->

- ⑥ config - , it is an object of ServletConfig.  
, It helps in passing the information to servlet or Jsp page during initialization  
, It is used to maintain configuration information about particular servlet.  
, getInitParameter()  
getServletName() etc.

The init parameters are in <init-param> of "web.xml".

Eg:

### web.xml

<web-app>

<servlet>

<servlet-name> MyDemo </servlet-name>

<jsp-file> /welcome.jsp </jsp-file>

```
<init-param>
 <param-name> username </param-name>
 <param-value> tiger </param-value>
</init-param>
</servlet>
<servlet-mapping>
 <servlet-name> MyDemo </servlet-name>
 <url-pattern> /demo </url-pattern>
</servlet-mapping>
</web-app>
```



In Jsp page it can be accessed as

- Welcome.jsp

```
<html>
 <body>
 <% String pname = config.getInitParameter("UserName");
 out.println(pname);
 %>
 </body>
</html>
```

- ⑧ session - , It is an object of HttpSession.  
, It is used to access the current client session.  
, getId(), getCreationTime(),  
setAttribute()  
getAttribute() etc.

Eg:

user.html

```
<html>
 <body>
 <form name="f1" action="hello.jsp">
 <input type="text" name="un">
 <input type="submit" value="click">
 </form>
 </body>
</html>
```

hello.jsp

```
<html>
 <body>
 <% String name = request.getParameter("un");
 out.println("welcome to " + name);
 session.setAttribute("user", name);
 %>
 Access
 </body>
</html>
```

getSession.jsp

```
<html>
```

```
 <body>
```

```
 <!String name = (String) session.getAttribute("user")>
```

```
 out.println("Session user is " + name);
```

```
 </>
```

```
 </body>
```

```
</html>.
```

⑦. pageContext - It is an instance of PageContext class. Used to represent the entire Jsp page.

/ It is used to share data between various Jsp pages based on its scope.

/ The method to store information is  
`pageContext.setAttribute("name", "value", "scope");`

/ The Scope can be

`pageContext.SESSION_SCOPE`,

`pageContext.SESSION_SCOPE`, (for session)

`pageContext.PAGE_SCOPE`; (for page)

`pageContext.APPLICATION_SCOPE` (for website)

`pageContext.REQUEST_SCOPE`. (for request)

/ To retrieve information

`pageContext.getAttribute("name", "scope");`

/ To remove attribute from the page

`pageContext.removeAttribute("name", "scope");`

User1.html

Ej:

```
<html>
 <body>
 <form name="fi" action="first.jsp">
 UserName : <input type="text" name="un">
 <input type="submit" value="click">
 </form>
 </body>
</html>
```

first.jsp

```
<html>
 <body>
 <% String name = request.getParameter("un");
 out.println("Welcome " + name);
 pageContext.setAttribute("user", name, pageContext.SESSION_SCOPE);
 %>
 Go to
 </body> </html>
```

second.jsp

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

```
<% String name = (String) pageContext.getAttribute("user", pageContext.SESSION_SCOPE);
 out.println("Hello " + name);
```

```
%>
```

```
</body>
```

```
</html>
```

9. application - It is an instance of ServletContext object.
- It is used to pass parameters to all JSP pages available in a web application (or) website.
  - This object is created when jsp page is initialized and removed when calls `JspDestroy()` method.
  - To set variable at application level,  
`application.setAttribute( String key, Object val);`
  - To get variable set at application level  
`application.getAttribute( String key);`

"Used for application level parameters".

Eg: User name in all webpages,  
no. of times a website visited. etc

Ej: // Count of no. of times visited website.

<html>

<body>

<%

Integer count = (Integer) application.getAttribute("hitCounter");

if (count == null || count == 0)

{

out.println("Welcome to site"); // first time

count = 1;

}

else

{

out.println("Welcome Back again");

count = count + 1;

}

application.setAttribute("hitCounter", count);

%>

<center>

<b> <h3> No. of visits is : </b> <= count </h3>

</h3> </b>

</center>

<body>

<html>

=====

### 3. Directive Elements

- Directive elements are used for providing special instructions to JSP container for processing the page.
- Provides directions to container
- The Syntax of directive element is  
`<%@ directive attribute = "value" %>`
- The directives can have several attributes.
- The attributes are key-value pairs and separated by commas.

The JSP directives are classified into 3 types.

- ① page directive
- ② include directive
- ③ taglib directive

#### 1. page directive

- Used to provide instructions to the container.
- The instructions are limited to page only.
- The syntax is  
`<%@ page attribute = "value" %>`
- It defines page-dependant attributes, scripting language, error page, buffering requirements etc.

The attributes of "page" directive are

- ① import — specifies a list of packages or classes used in JSP page.

Eg: <%@ page import = "java.util.\*" %>  
<%@ page import = "java.util.Date" %>  
<%@ page import = "java.io.\*,  
javax.servlet.\*,  
java.sql.\*" %>

- ② session — Specifies whether or not the JSP page participates in HTTP session.  
The possible values are true/false.

<%@ page session = "true/false" %>

- ③ errorPage — Defines the URL of another JSP that reports on Java unchecked runtime exceptions.  
i.e. used to handle exceptions.

<%@ page errorPage = "errorhandler.jsp" %>

- ④ isErrorPage → It is used to check whether a particular JSP page will act as a exception handler or not. (true/false)

<%@ page isErrorPage = "true/false" %>

⑤ extends - Specifies the super class that the generated servlet must extend.  
i.e. current page acquires properties of other class.

<!--@ page extends = "className" -->

⑥ language - Defines the programming language used in the JSP page.

<!--@ page language = "java" -->

⑦ buffer - It is used to specify the output buffer size. By default it is 8KB.

<!--@ page buffer = "16KB" -->

⑧ contentType - Used to specify the type of response formulated by the JSP container.

<!--@ page contentType = "text/html" -->



text/css

text/html

image/gif etc.

Eg. <!--@ page import = "java.util.\*" contentType = "text/css"  
session = "true" errorPage = "handler-jsp" -->

## 2. include directive

- Includes a file during the translation in current page.
- It is similar to `<jsp:include>` action.
- include directive gets the content at the time of translation, whereas `<include>` action gets at the time of processing request

`<%@ include file = "url" %>`

Eg: `<%@ include file = "welcome.jsp" %>`

## 3. taglib directive

- Declares a tag library, containing custom actions used in the page.
- JSP allows to define custom JSP tags, those are declared using taglib directive.
- The Syntax is

`<%@ taglib uri = "taglibraryname"  
prefix = "name" %>`

## Deploying Java Bean In a JSP Page

- Java Beans are reusable components , helps us keeping the business logic separate from presentation logic.
  - Beans are used in Jsp pages as the instance of class.
  - The scope of the bean must be Specified .  
The various scopes using which a bean can be used in Jsp page are
- ① Page scope - The default scope for a bean in JSP page is "page scope". The bean object gets disappeared as soon as the current page get discarded.
  - ② Request scope - The bean object remains in existence as long as the request object is present.
  - ③ Session scope - A session can be defined as the specific period of a time the user spends browsing the site . i.e. A session is Time Interval.
  - ④ Application scope - During application scope the bean will get stored to ServletContext. Hence the bean is available to all servlets in the same Web application.

✓ `<jsp:useBean>` is used to specify the scope of bean.

Eg: `<jsp:useBean id = "ID102" class = "counter.Count1"  
scope = "application" />`

## Example - 1

Date  
Pages

Step 1 :- Creating a Simple bean in a package.

```
package mypack;
public class MessageDemo
{
 public static String msg;
 public MessageDemo()
 {
 msg = "Welcome to Bean";
 }
 public String getMsg()
 {
 return (msg);
 }
 public void setMsg(String m)
 {
 msg = m;
 }
}
```

MessageDemo.java

Step 2: Creating a JSP page to access Bean.

```
<html>
```

```
<body>
```

```
<jsp:useBean id="mb" scope="Session"
 class = "mypack.MessageDemo" >
```

```
<jsp: getProperty name = "mb" property = "msg" />
<jsp: setProperty name = "mb" property = "msg"
value = "Hi Hyderabad" />
</jsp:useBean>
<jsp: getProperty name = "mb" property = "msg" />
</body>
</html>.
```

### Example - 2

"Sharing Session & Application data with example".

Step - 1 :- Creating bean class in a package.

```
package count;
public class CountDemo
{
 public int count;
 public CountDemo()
 {
 count = 0;
 }
 public int getCount()
 {
 return count;
 }
 public void setCount (int x)
 {
 count = x;
 }
}
```

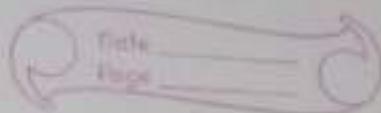
Step 2 : Accessing Bean with "Session" Scope.

```
<html>
 <head>
 <title> Session scope </title>
 </head>
 <body>
 <jsp:useBean id="c1" scope="session"
 class = "count.CountDemo" />
 <% c1.setCount(c1.getCount() + 1); %>
```

The Count is

```
<jsp:getProperty name="c1" property="count" />
<% = c1.getCount() %>
</body>
</html>
```

Step-3 :- " If we open the JSP page in two different browsers, each is a separate session and count is incremented independently each time".



Step 4 :- Accessing bean with "application" scope.

```
<html>
```

```
<head>
```

```
<title> Application scope </title>
```

```
</head>
```

```
<body>
```

```
<jsp:useBean id="c1" class="count.CountDemo"
scope = "application" />
```

```
<% c1.setCount(c1.getCount() + 1); %>
```

```
<h1> The count is </h1>
```

```
<%= c1.getCount() %>
```

```
</body>
```

```
</html>
```

Steps : If we access Jsp page in two different browsers, for each request count is incremented in synchronized manner.

Eg: If count = 10 with Mozilla

If it access the page with Google chrome,

the count = 11.

Both browser use same.

# Accessing Database from a JSP Page

(empdata.jsp)

① // Retrieving Employees Data from Emp table.

```
<%@ page language = "java" import = "java.sql.*" %>
<%@ page import = "java.io.*" %>
<%
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/employee", "root", "");
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("select * from emp");
%>
```

<html>

<body>

<center>

<H2> Employees Data is </H2>

<table border = "3">

<tr>

<th> EmpID </th>

<th> EmpName </th>

<th> Department </th>

<th> Salary </th>

</tr>

P.T.O.

<%

while (rs.next())

{

</%>

<tr>

<td> <%= rs.getInt(1) %> </td>

<td> <%= rs.getString(2) %> </td>

<td> <%= rs.getString("dept") %> </td>

<td> <%= rs.getInt(4) %> </td>

</tr>

<%

}

con.close();

%>

</center>

</body>

</html> .

## // Inserting Data into emp table - using JSP ( insert.jsp ).

```
/* HTML page to insert data */
<html>
 <body>
 <form name = "f1" method = "post"
 action = "http://localhost:8080/myjsp3/
 empinsert.jsp">

 <center>
 EmpId : <input type = "text" name = "eid"
 size = 30 >

 EmpName : <input type = "text" name = "ename"
 size = 30 >

 Dept : <input type = "text" name = "edept"
 size = 30 >

 Salary : <input type = "text" name = "esal"
 size = 30 >

 <input type = "submit" value = "Insert" >
 </center>
 </form>
</body>
</html>
```

Date \_\_\_\_\_  
Page \_\_\_\_\_

```
/* Inserting and accessing using JSP */
<%@ page language="java" import="java.sql.*" %>
<%@ page import="java.io.*" %>
<%
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection(
 "jdbc:mysql://localhost:3306/employee", "root", "");
Statement st = con.createStatement();
int x1 = Integer.parseInt(request.getParameter("eid"));
String x2 = request.getParameter("ename");
String x3 = request.getParameter("edept");
int x4 = Integer.parseInt(request.getParameter("esal"));

st.executeUpdate(" INSERT INTO emp(id, name,
dept, salary) VALUES
(' "+x1+" ',' "+x2+" ', '"+x3+" ',
"+x4+" ')");
%>
```

```
ResultSet rs = st.executeQuery(
 " select * from emp");
```

>

```
<html>
 <body>
 <center>
 <h2> The Employee data is </h2>
 <table border = "3">
 <tr>
 <th> EmpID </th>
 <th> EmployeeName </th>
 <th> Department </th>
 <th> Salary </th>
 </tr>
 <%
 while (rs.next())
 %>
 <tr>
 <td> <.= rs.getInt(1) > </td>
 <td> <.= rs.getString(2) > </td>
 <td> <.= rs.getString(3) > </td>
 <td> <.= rs.getInt(4) > </td>
 </tr>
 <%
 con.close();
 </center> </body>
 </html>
```

## (3) /\* Use of Prepared Statement using Jsp \*/

&lt;--/0

====

```
int x1 = Integer.parseInt(request.getParameter("eid"));
String x2 = request.getParameter("ename");
String x3 = request.getParameter("edept");
int x4 = Integer.parseInt(request.getParameter("esal"));
→ String query = "INSERT INTO emp(id, name, dept, salary) VALUES
→ Prepared Statement st =
con.prepareStatement(query);
```

```
st.setInt(1, x1);
```

```
st.setString(2, x2);
```

```
st.setString(3, x3);
```

```
st.setInt(4, x4);
```

```
st.executeUpdate();
```

```
ResultSet rs = st.executeQuery("select * from emp");
```

&lt;--/0

====

=====

# Error Handling & Debugging

Date \_\_\_\_\_  
Page \_\_\_\_\_

During JSP code, we can have following type of errors.

① Checked Exceptions - The exceptions that are notified at the time of compilation.

Eg: FileNotFoundException, ClassNotFoundException

② Runtime Exceptions - These are ignored at the time of compilation. It is an exception, that

Division by zero

{ probably could have been avoided by the programmer}

③ Errors - Problems that arise/ arise beyond the control of the user program. Ignored by compiler.

Eg: Stack overflow.

ArithmeticException, ArrayIndexOutOfBoundsException,  
NullPointerException etc.

JSP handles Runtime Exceptions.

JSP uses predefined object exception which is an instance of a subclass of Throwable. The methods are

public String getMessage() - A detailed message about the exception that has occurred.

public Throwable getCause() - Returns the cause of the exception.

public void printStackTrace() - Displays the error path.

Ex:

// Example Program to Handle Exception.

```
"main.jsp"
<%@ page errorPage = "showError.jsp" %>
<html>
 <head>
 <title> Error Handling </title>
 </head>
 <body>
 <%
 int x = 1;
 if (x == 1)
 throw new RuntimeException("Error");
 %>
 </body>
</html>

showError.jsp
<%@ page isErrorPage = "true" %>
<html>
 <body>
 <p> Sorry Error Occurred </p>
 <% exception.printStackTrace(response.getWriter()); %>
 </body>
</html>
```

// Program to handle ArithmeticException

```
<html>
 <body>
 <form name="f1" action="perform.jsp">
 Num1 : <input type="text" name="t1">
 Num2 : <input type="text" name="t2">
 <input type="submit" value="find">
 </form>
 </body>
</html>
```

→ input.html

→ perform.jsp

```
<html>
 <body>
 <%@ page errorPage = "error.jsp" %>
 <% int n1 = Integer.parseInt(request.
 getParameter("t1"));
 int n2 = Integer.parseInt(request.getParameter("t2"));
 int c = n1/n2;
 out.println("C is " + c);
 %>
```

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

P-I-O

error.jsp

<html>

<body>

<%@ page isErrorPage = "true" %>

Exception information

<% = exception %>

</body>

</html>

## Sharing Data B/w JSP Pages

- ① <jsp:forward> with <jsp:param>
- ② pageContext object
- ③ sharing session & application data  
using session & application object

## UNIT - V

Topic  
Page

### Database Access Using Java (JDBC)

JDBC - (Java Database Connectivity)

JDBC is a specification from Sun Microsystems that provides a standard API & protocols for Java applications to communicate with different databases.

JDBC is used to write programs required to access databases.

JDBC & database driver, is capable of accessing databases.

JDBC is a platform independent API (interface) b/w RDBMS & Java programming language.

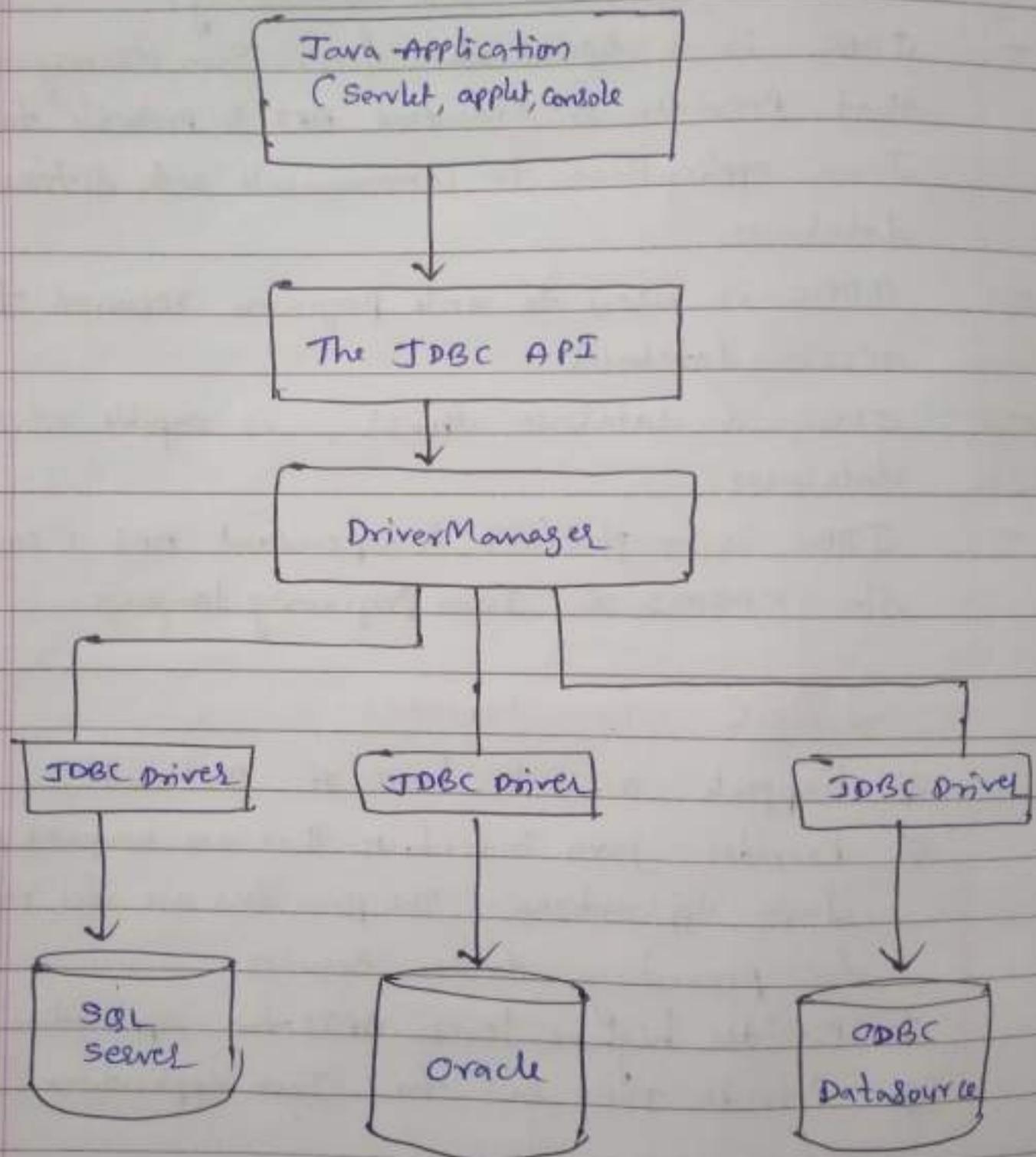
### JDBC Characteristics

- \* Supports a wide level of portability
- \* Provides java interfaces that are compatible with Java applications. The providers are also responsible for providing driver services.
- \* Provides higher level APIs for application programmers.
- \* Provides JDBC API for Java applications.

### Components of JDBC

1. JDBC API
2. JDBC Driver Manager
3. JDBC test suit
4. JDBC - ODBC Bridge.

## JDBC Architecture



- A JDBC driver is required to process the SQL requests and generate results.
- JDBC API provides classes and interface to handle database specific calls from user.

In the above diagram

- ① The Java application that needs to communicate with database has to be programmed using JDBC API.
- ② The JDBC driver (third-party vendor) supporting datasource like oracle, sql is added in the java application for JDBC support.

\* \* \* - To communicate with any datasource through JDBC, we need a JDBC driver that intelligently communicates with datasource.

## Types of Drivers

Date  
Page

In JDBC 4 types of Drivers are available.

- ① Type-1 Driver → JDBC-ODBC bridge driver
- ② Type-2 Driver → Native-API partly Java driver
- ③ Type-3 Driver → Pure Java driver for  
(Network protocol driver)  
← (Uses a middleware driver to connect to DB).
- ④ Type-4 Driver → pure Java driver (Thin driver / Full Java)  
which is directly connected to a database  
(Thin driver / Full Java)

### ① Type-1 Driver (JDBC-ODBC Bridge Driver).

- Acts as a bridge between JDBC and other database connectivity mechanisms. (like ODBC).
- The JDBC-ODBC driver (bridge driver) provided by Sun is example of Type-1 driver.
- Used to communicate with existing data sources.
- This driver converts JDBC calls into ODBC calls and redirects the request to ODBC driver.
- . The Type-1 driver process is

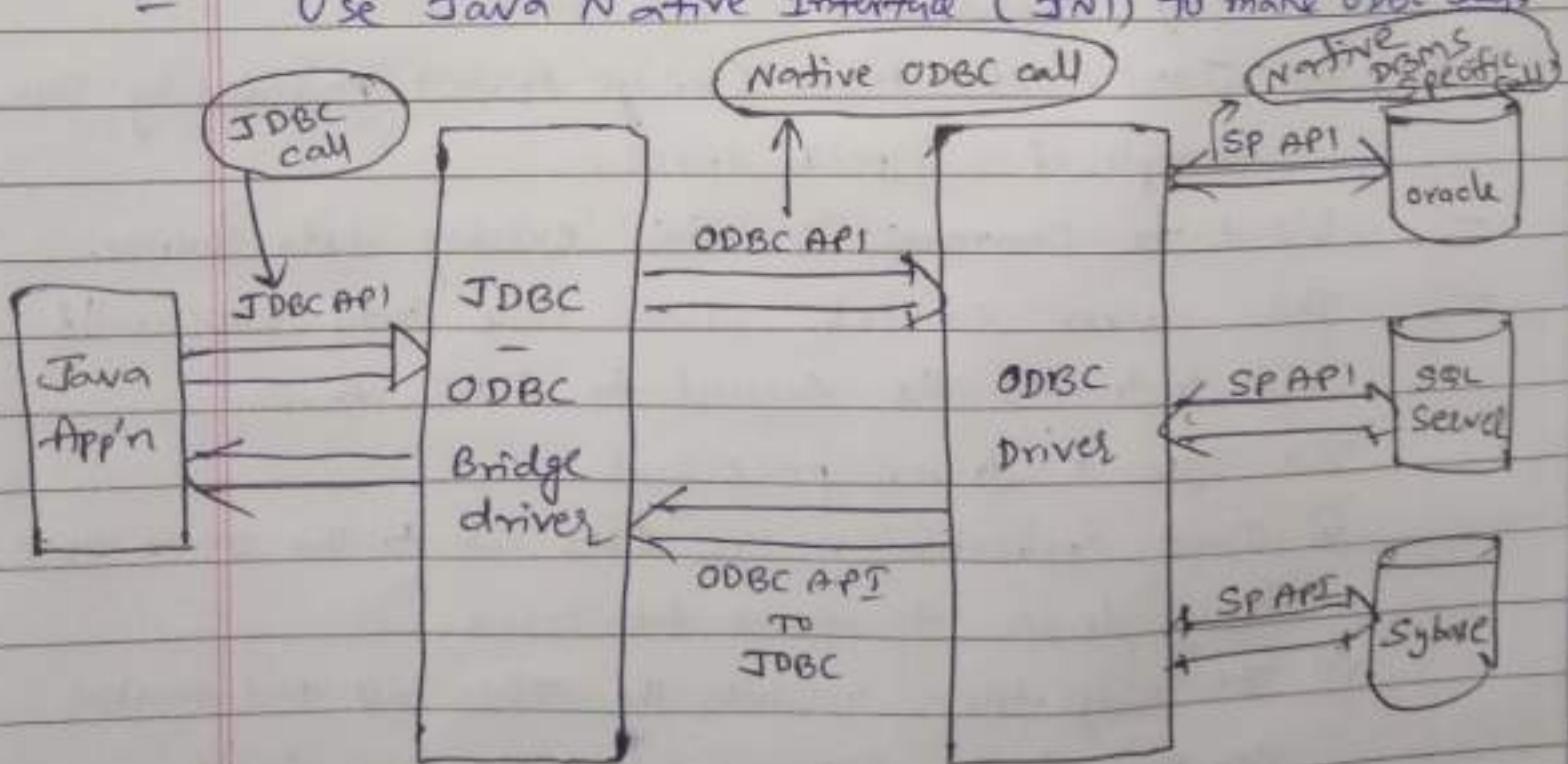
- ① Java application makes JDBC call to the JDBC-ODBC bridge driver to access data source.
- ② The bridge driver resolves the JDBC call and makes an equivalent ODBC call to the ODBC driver.
- ③ ODBC driver completes the request and sends response to bridge driver.
- ④ The driver converts the response into JDBC standard and display the result to Java application.

## Advantages

- only one driver implementation to interact with different data sources.
- Communicating with all databases supported by ODBC driver.
- A vendor independent driver.

## Dis-Advantages

- Decrease execution speed, due to large no. of translation b/w JDBC & ODBC.
- Depends on ODBC driver.
- ODBC client library installed in every client.
- Use Java Native Interface (JNI) to make ODBC calls.

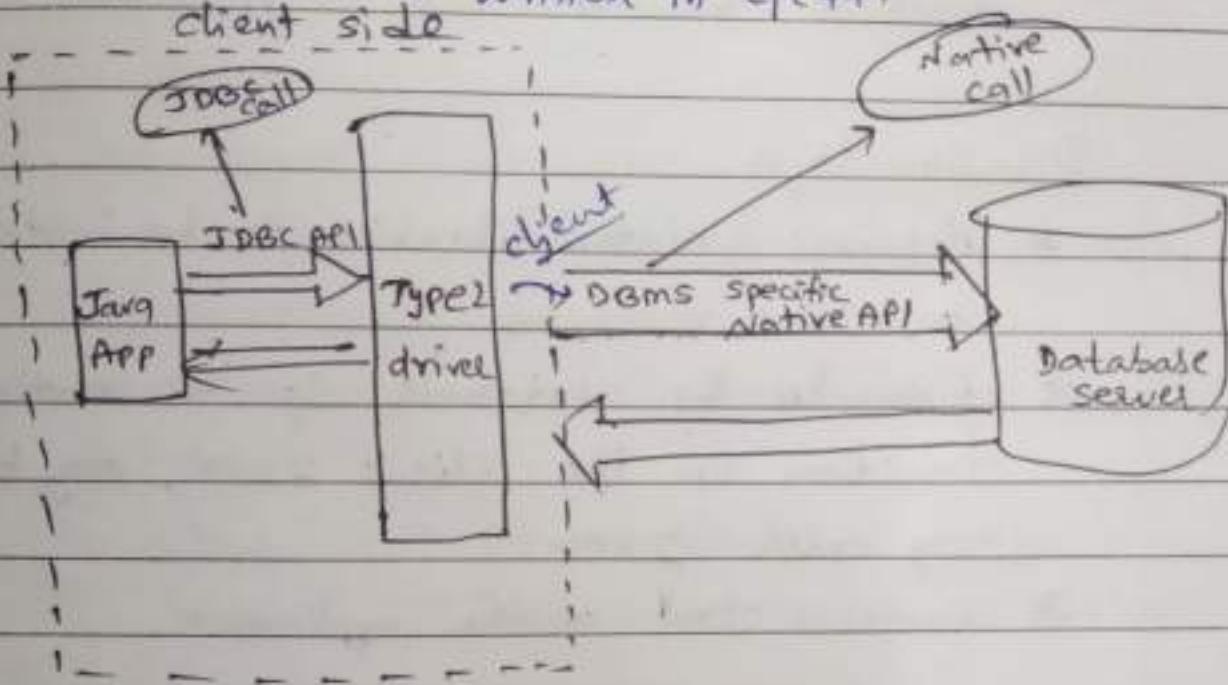


2

## Type - 2 Driver (Native - API driver)

The JDBC call can be converted into the database vendor specific native call with the help of Type-2 driver. called Native API Driver

- Native API is written in c/c++.



The Process is

- ① Java application that needs to communicate with database is written using JDBC API.
- ② JDBC calls are converted into database specific native calls in the client machine and the request is dispatched to database specific native libraries. (using native protocols).

\* Type-2 drivers used with server side applications only. Bcz the database specific native libraries should be installed on the client machines. This type of driver is implemented for a specific database and delivered by DBMS vendor.

## Advantages

- \* Access data faster as compared to other types of drivers
- \* Additional features provided by specific database vendor also supported.

## Disadvantages

- \* Requires Native libraries to be installed on client machine
- \* Executes the database specific native functions on the client JVM, any bug may crash JVM.
- \* Increase Cost of the application.

## Type-3 Drivers (Network Protocol Driver)

- Uses middleware (Application server) that converts JDBC calls to vendor specific database protocol.
  - It is fully written in Java.  
The process is
    - ① The driver listens for JDBC calls from the Java application and translates them into middleware specific calls.
    - ② The middleware server converts these calls into database specific calls.
  - Also called net-protocol drivers.
  - The middleware has pool management, performance improvement & connection availability features.
  - Uses socket communication.
- \* \* Type-3 driver is more useful in Enterprise applications use with applets directly.

### Advantages

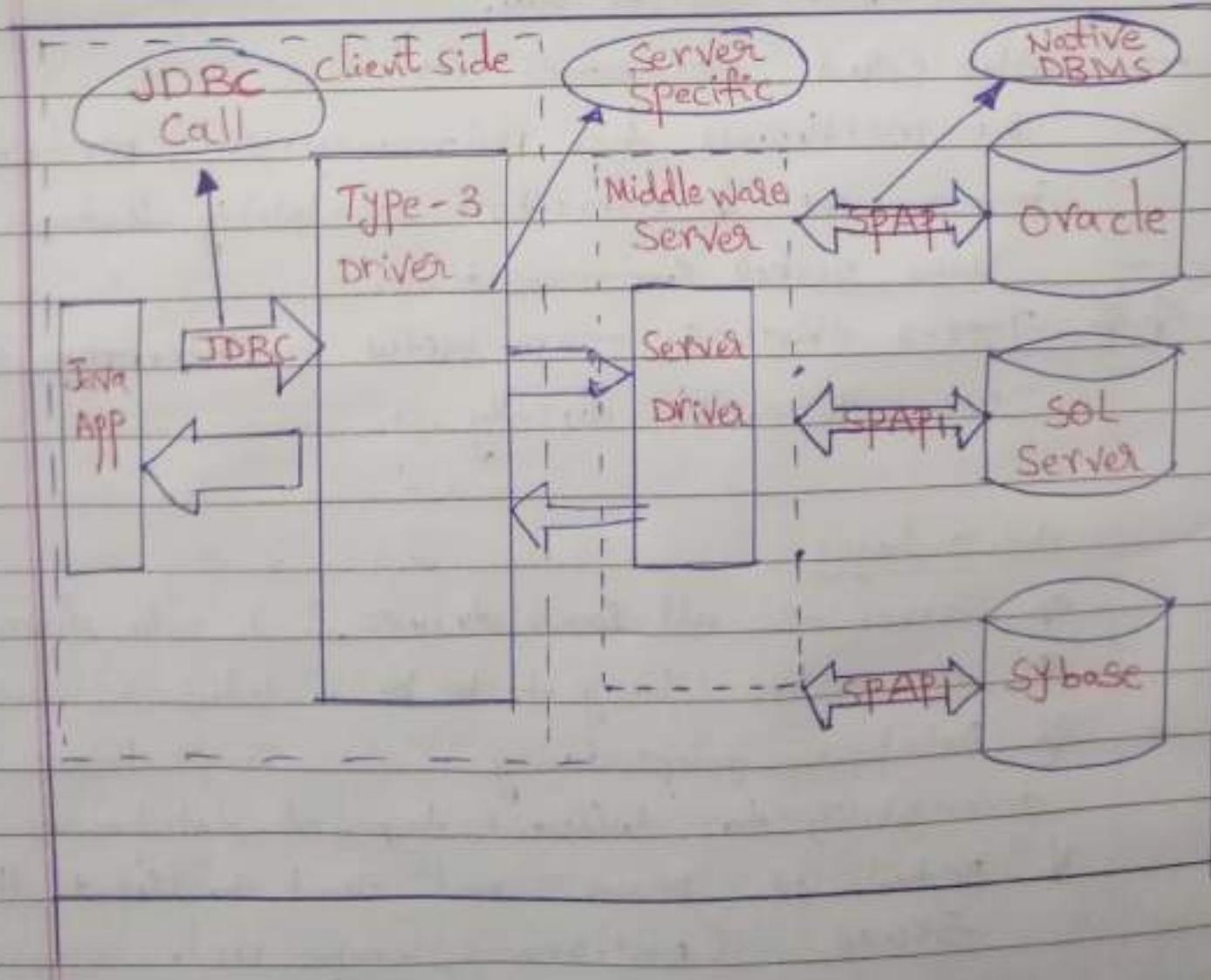
- \* Serves as all Java driver & is auto downloadable
- \* No Native library to be installed on client machine
- \* Database independency . i.e. Single driver provides accessibility to different type of databases.
- \* Details of DBMS not sent to client. Hence secure. (pwd, uname, location etc).

\* Provides the facility to switch from one database to another, without change in client-side driver classes.

Just need to change middleware server.

## Disadvantages

- Network support is needed on client machine
- Performs the task slowly due to increased number of network calls
- Costlier than other drivers.



## Type-4 Driver (Thin driver)

Also called Java to Database protocol.

It is a pure Java driver, does not require any native database library to retrieve the records from database.

The driver implements database protocol to interact directly with a database.

It translates JDBC calls into database specific N/W calls.

The process is

① Type-4 driver prepares a DBMS Specific N/W message and then communicates with database server over a socket.

- It is a lightweight and generally known as "thin driver".

- uses database specific proprietary protocol for communication.

- Generally implemented by ~~Type-4 driver~~ DBMS vendor.

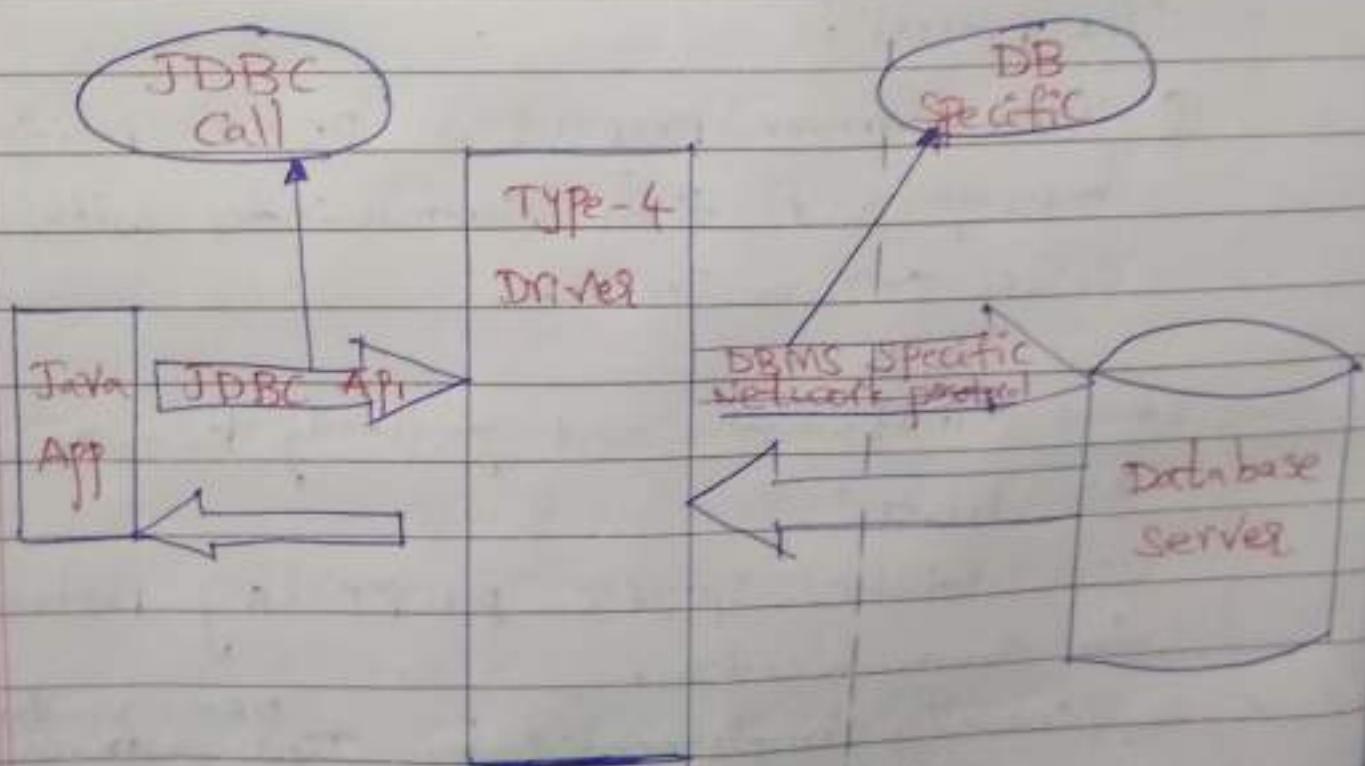
Its Advantages

uses database specific proprietary protocol and is DBMS vendor dependant.

## Advantages

- ✓ Serves as a pure Java driver and is auto up downloadable.
- ✓ does not require any native library to be installed on client machine.
- ✓ Uses database server specific protocols.
- ✓ Does not require middleware server.

client side



# Procedure for Accessing a Database from Java API provided in "Java.sql.\*".

## 1. Registering a JDBC driver with Driver Manager

- Before using any driver, it must be registered with the Driver Manager.
- All drivers contain built-in class name.
- To register a driver, we need to load a driver into client application.
- forName() method of 'Class' is used to register the driver class.
- This method is used to dynamically load the driver class.

```
public static void forName(String classname)
throws ClassNotFoundException
```

Eg: for Type 1 driver

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

class name, it is same  
for any database

JdbcOdbcDriver

for Type 4 driver

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

for MySQL database

```
Class.forName("com.mysql.jdbc.Driver");
```

## 2. Establish a Connection with database. (Creating Connection object)

- After loading a driver, DriverManager is responsible to establish a connection with database.
- `getConnection()` method of DriverManager class is used to establish connection with the database.

```
public static Connection getConnection(String url)
 throws SQLException
```

```
public static Connection getConnection(String url,
 String name, String password)
 throws SQLException
```

Connection con = DriverManager.getConnection(url, username, password)

Eg: url → specifies the location of database.

url format → jdbc : sub-protocol : datasourceName

for Type 2 driver ⇒ jdbc : odbc : dbname

for Type 4 driver ⇒ "jdbc : oracle : thin : @localhost : 1521 : dbname"

Port no - 1521 for oracle

Port no - 3306 for mysql

```
Connection con = DriverManager.getConnection
("jdbc : oracle : thin : @localhost : 1521 : xe",
"system", "password");
```

for Mysql

```
Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/mydb", "root", "");
```

By default password is Empty for mysql.

### 3. Creating SQL Statement. (Creating statement object)

To perform any operation on database, need to Create Suitable SQL Statement.

The Three types of Statements are

- ① Statement
- ② PreparedStatement
- ③ CallableStatement

Q: Statement st = con.createStatement();

The createStatement() of Connection interface is used to create statement.

Public Statement ~~at~~ createStatement() throws  
SQLException.

X [To execute an SQL Statement]

#### 4. Execute the SQL statement (Execute Query).

After creating object of Statement, the query is executed.

To execute SQL statements the following functions are used, provided by Statement interface.

- (i) `executeQuery()` - Used only with "select" Command, it returns the result as a ResultSet object, contains rows/records of a table.
- (ii) `ResultSet rs = st.executeQuery("Select * from emp");`  
↳ `executeUpdate()` - Used with insert, delete, update or create statement. It returns integer value, represents no. of rows effected.

Ex: `int x = st.executeUpdate("Insert into emp values (1093, "Ramesh", "VCE")");`

- (iii) `execute()` - Used with all SQL commands. It is a boolean function. Returns status.

Ex: `boolean x = st.execute(query);`

To retrieve result set, `getResultSet()` is used.

## 5. Process the results.

To retrieve from ResultSet the following methods are used.

next() - Returns true if next record is available

previous() - Returns true if previous record is available

first() - Returns first record.

To access particular field getter methods are used.

type getType(); || getXXX() methods

int getInt("eno");

String getString("ename");

type getXXX(field no)

(or)

type getXXX(field name).

Ex: ResultSet rs = st.executeQuery("select \* from emp");  
while (rs.next())

{

    rs.getInt  
    ↓ ("eno") ;

    System.out.println(rs.getString("ename"));  
    System.out.println(rs.getString("Cname"));

}

P.T.O.

## 6. Closing the Connection.

public void close() throws SQLException  
Once data retrieved / accessed, the connection  
to datasource is released using close().

rs.close(); }  
st.close(); } (or) only  
con.close();

---

## java.sql.\*

- It provides API for accessing and processing data stored in database source (RDBMS) using java application.
- The package contains classes and interfaces for JDBC API.

### The commonly used interfaces

S.No	Interface Name	Description
1	<b>Driver</b>	Every driver class must implement to load driver and work with database
	<b>Connection</b>	A connection (session) with a specific database. SQL statements are executed and results are returned within the context of a connection.
The methods of the interface are		
2	<code>void close()</code>	Releases this Connection object's database and JDBC resources immediately
	<code>void commit()</code>	Makes all changes permanent
	<code>boolean isClosed()</code>	Retrieves whether this Connection object has been closed.
	<code>boolean isReadOnly()</code>	Retrieves whether this Connection object is in read-only mode.
	<code>void rollback()</code>	Undoes all changes made in the current transaction and releases any database locks currently held by this Connection object.
	<code>Statement createStatement()</code>	Creates a Statement object for sending SQL statements to the database.
	<code>PreparedStatement prepareStatement(String sql)</code>	Creates a PreparedStatement object for sending parameterized SQL statements to the database.
	<code>CallableStatement prepareCall(String sql)</code>	Creates a CallableStatement object for calling database stored procedures.
	<b>Statement</b>	The object used for executing a static SQL statement and returning the results it produces.
The methods of the interface are		
3	<code>void close()</code>	Releases this Statement object's database and JDBC resources immediately
	<code>boolean execute(String sql)</code>	Executes the given SQL statement, which may return multiple results.
	<code>int[] executeBatch()</code>	Submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts.
	<code>ResultSet executeQuery(String sql)</code>	Executes the given SQL statement, which returns a single ResultSet object.
	<code>int executeUpdate(String sql)</code>	Executes the given SQL statement, which may be an INSERT, UPDATE, or DELETE statement or an SQL

		statement that returns nothing, such as an SQL DDL statement.
	<b>Connection getConnection()</b>	Retrieves the Connection object that produced this Statement object.
	<b>int getMaxRows()</b>	Retrieves the maximum number of rows that a ResultSet object produced by this Statement object can contain.
	<b>boolean isClosed()</b>	Retrieves whether this Statement object has been closed.
4	<b>PreparedStatement</b>	An object that represents a precompiled SQL statement. A SQL statement is precompiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.
	The methods of the interface are	
	<b>boolean execute()</b>	Executes the SQL statement in this PreparedStatement object, which may be any kind of SQL statement.
	<b>ResultSet executeQuery()</b>	Executes the SQL query in this PreparedStatement object and returns the ResultSet object generated by the query.
	<b>int executeUpdate()</b>	Executes the SQL statement in this PreparedStatement object, which must be an SQL Data Manipulation Language (DML) statement, such as INSERT, UPDATE or DELETE; or an SQL statement that returns nothing, such as a DDL statement.
	<b>void setByte(int parameterIndex, byte x)</b>	Sets the designated parameter to the given Java byte value.
	<b>void setDouble(int parameterIndex, double x)</b>	Sets the designated parameter to the given Java double value.
	<b>Void setFloat(int parameterIndex, float x)</b>	Sets the designated parameter to the given Java float value.
	<b>void setInt(int parameterIndex, int x)</b>	Sets the designated parameter to the given Java int value.
	<b>void setLong(int parameterIndex, long x)</b>	Sets the designated parameter to the given Java long value.
	<b>void setString(int parameterIndex, String x)</b>	Sets the designated parameter to the given Java String value.
	<b>ResultSet</b>	A table of data representing a database result set, which is usually generated by executing a statement that queries the database.
The methods of the interface are		
	<b>Void close()</b>	Releases this ResultSet object's database and JDBC resources immediately instead of waiting for this to

		happen when it is automatically closed.
5	<code>Void deleteRow()</code>	Deletes the current row from this ResultSet object and from the underlying database.
	<code>Boolean first()</code>	Moves the cursor to the first row in this ResultSet object.
	<code>Boolean getBoolean(int columnIndex)</code>	Retrieves the value of the designated column in the current row of this ResultSet object as a boolean .
	<code>Boolean getBoolean(String columnLabel)</code>	Retrieves the value of the designated column in the current row of this ResultSet object as a boolean.
	<code>Byte getByte(int columnIndex)</code>	Retrieves the value of the designated column in the current row of this ResultSet object as a byte.
	<code>Byte getByte(String columnLabel )</code>	Retrieves the value of the designated column in the current row of this ResultSet object as a byte
	Similarly	
	<code>getDate(int columnIndex), getDate(String columnLabel) getDouble(int columnIndex)</code>	
	<code>getDouble(String columnLabel)</code>	
	<code>getFloat(int columnIndex)</code>	
	<code>getFloat(String columnLabel)</code>	
	<code>getInt(int columnIndex)</code>	
	<code>getInt(String columnLabel)</code>	
	<code>getLong(int columnIndex)</code>	
	<code>getLong(String columnLabel)</code>	
	<code>getShort(int columnIndex)</code>	
	<code>getShort(String columnLabel)</code>	
	<code>getString(int columnIndex)</code>	
	<code>getString(String columnLabel)</code>	
6	<b>CallableStatement</b>	The interface used to execute SQL stored procedures.
	The methods of the interface are	
	<code>byte getByte(int parameterIndex)</code>	Retrieves the value of the designated parameter as a byte.
	<code>Byte getByte(String parameterName)</code>	Retrieves the value of a parameter as a byte.
	Similarly for int,short,float,string,long and double,boolean	

	<code>getInt(),getShort(), getFloat(), getLong(), getDouble(),getString(),getBoolean()</code>	
	<code>setByte(String parameterName, byte x)</code>	Sets the designated parameter to the given Java byte value.
Similarly for int,short,float,string,long and double,Boolean <code>setInt() , setShort(),setLong(),setFloat(),setDouble(),setString(),setBoolean()</code> etc		
7	<b>ResultSetMetaData</b>	An object that can be used to get information about the types and properties of the columns in a ResultSet object
	<b>Int getColumnCount()</b>	Returns the number of columns in this ResultSet object.
	<b>String getColumnName(int column)</b>	Get the designated column's name
	<b>String getColumnTypeName(int column)</b>	Retrieves the designated column's database-specific type name.
	<b>String getTableName(int column)</b>	Gets the designated column's table name.
	<b>Boolean isReadOnly(int column)</b>	Indicates whether the designated column is definitely not writable.
	<p>Similarly</p> <pre><b>isCaseSensitive</b>(int column) <b>isNullable</b>(int column) <b>isWritable</b>(int column)</pre>	

The Class used for Connection is DriverManager.

7	<b>DriverManager</b>	The basic service for managing a set of JDBC drivers.
	The Methods defined are	
	<b>static Connection getConnection(String url)</b>	Attempts to establish a connection to the given database URL.
	<b>Static Connection public static Connection getConnection(String url, String user, String password)</b>	Attempts to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.
	<b>static Driver getDriver(String url)</b>	Attempts to locate a driver that understands the given URL.
	<b>static void println(String message)</b>	Prints a message to the current JDBC log stream.

## PreparedStatement

- ✓ Prepared Statement queries are pre-compiled on database and there access plan will be reused to execute further queries which allows them to execute much quicker than normal queries generated by Statement object.
- ✓ The PreparedStatement interface is a sub interface of Statement. It is used to execute parameterized query.
- ✓ We are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.
- ✓ "?" is also called placeholder or IN parameter in Java.
- ✓ Index of placeholder or parameter starts with "1" and not with "0".
- ✓ Allows writing dynamic and parametric query.
- ✓ PreparedStatement is faster than Statement in Java

The **prepareStatement()** method of **Connection** interface is used to return the object of PreparedStatement

**public PreparedStatement prepareStatement(String query)**

### **Methods of PreparedStatement interface**

Method	Description
<b>public void setInt(int paramInt, int value)</b>	sets the integer value to the given parameter index.
<b>public void setString(int paramInt, String value)</b>	sets the String value to the given parameter index.
<b>public void setFloat(int paramInt, float value)</b>	sets the float value to the given parameter index.
<b>public void setDouble(int paramInt, double value)</b>	sets the double value to the given parameter index.
<b>public int executeUpdate()</b>	executes the query. It is used for create, drop, insert, update, delete etc.
<b>public ResultSet executeQuery()</b>	executes the select query. It returns an instance of ResultSet.

```

//Insert records using PreparedStatement
import java.sql.*;
import java.util.*;

public class PrepareTable
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee","root","");
 PreparedStatement st=con.prepareStatement("INSERT INTO emp VALUES(?,?,?,?,?)");

 Scanner sc = new Scanner(System.in);
 System.out.println("Enter Emp Id");
 int id1= sc.nextInt();
 System.out.println("Enter Emp Name");
 String na = sc.next();
 System.out.println("Enter Emp Dept");
 String dp = sc.next();
 System.out.println("Enter Emp Salary");
 int sa = sc.nextInt();

 st.setInt(1,id1);
 st.setString(2,na);
 st.setString(3,dp);
 st.setInt(4,sa);

 int i=st.executeUpdate();
 System.out.println(i+" records inserted");

 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 System.out.println("Row Inserted");
 }
}

```

```

//Retrieve records based on parameter using PreparedStatement
import java.sql.*;
import java.util.*;

public class PrepareTable1
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee","root
","");
PreparedStatement st=con.prepareStatement("SELECT * from emp where
dept=?");

Scanner sc = new Scanner(System.in);
System.out.println("Enter Department Name");
String dpt= sc.next();

st.setString(1,dpt); //st.setString(1,'cse');
ResultSet rs = st.executeQuery();

System.out.println("The Employee Details are \n");
while(rs.next())
{
 System.out.println(rs.getInt(1)+":"+rs.getString(2)+":"
+rs.getString(3)+":"+rs.getInt("salary"));
 System.out.println();
 //rs.getInt("id");
 //rs.getString("name");
}

con.close();
}
catch(Exception e)
{
 System.out.println(e);
}
System.out.println("Row Inserted");
 }
}

```

### **//Update records based on parameter using PreparedStatement**

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");

stmt.setString(1,"Vignesh");//1 specifies the first parameter in the query i.e. name
stmt.setInt(2,346);

int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

### **//retrieve records using PreparedStatement**

```
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next())
{
 System.out.println(rs.getInt(1)+":"+rs.getString(2)+":"+
 rs.getString(3)+":"+rs.getInt("salary"));
}
```

### **//delete records using PreparedStatement**

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
stmt.setInt(1,101);

int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
```

## CallableStatement

- ✓ CallableStatement interface is used to call the stored procedures and functions.
- ✓ We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.
- ✓ The differences between stored procedures and functions are given below:

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

The `prepareCall()` method of Connection interface returns the instance of CallableStatement.

**Syntax:**

```
public CallableStatement prepareCall("{ call procedurename(?,?...?) }");
```

```
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
```

**The syntax of procedure**

```
CREATE procedure proc-name
(type IN name , type IN name , .. type OUT name)
```

```
BEGIN
```

```
// Query to be executed
END
```

```
Eg: CREATE procedure add
(int IN x , int IN y , int OUT z)
BEGIN
 Z=x+y;
END
```

```
//CallableStatement Demo - insertion - with Procedure Call
//goto database - select routines - add routine - insert arguments - write
query - submit.
import java.sql.*;
import java.util.*;

public class CallableDemo
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee","root","");

CallableStatement st=con.prepareCall("{call INSERTDATA(?,?,?,?,?)}");

Scanner sc = new Scanner(System.in);
System.out.println("Enter Emp Id");
int id1= sc.nextInt();
System.out.println("Enter Emp Name");
String na = sc.next();
System.out.println("Enter Emp Dept");
String dp = sc.next();
System.out.println("Enter Emp Salary");
int sa = sc.nextInt();

st.setInt(1,id1);
st.setString(2,na);
st.setString(3,dp);
st.setInt(4,sa);

st.execute();

con.close();
```

```
 }
 catch(Exception e)
 {
 System.out.println(e);
 }

 System.out.println("Row Inserted");
 }
}
```

### //retrieving data using CallableStatement

Inside procedure GETDATA

---

Select \* from emp

```
CallableStatement st = con.prepareCall("{call GETDATA}");
ResultSet rs = st.executeQuery();
```

### //delete data using CallableStatement

Inside procedure DELETEDATA (IN x)

delete from emp where id=x

```
CallableStatement st = con.prepareCall("{call DELETEDAT(153)}");
St.execute();
```

### //update data using CallableStatement

Inside procedure UPDATEDATA (IN x , IN y)

update emp set name=x where id=y

```
CallableStatement st = con.prepareCall("{call UPDATEDATA(?,?)}");

St.setString(1,"Ramesh");
St.setInt(2,153);

St.execute();
```

## JDBC Programs

```
/* The changes to do in UBUNTU execution
port no-3307 ,
Class.forName("com.mysql.cj.jdbc.Driver"); */

//Create Table USING JDBC - MySQL
import java.sql.*;
public class CreateTable
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection con=DriverManager.getConnection(
 "jdbc:mysql://localhost:3306/employee","root","");
 Statement st=con.createStatement();

 st.executeUpdate("create table emp (id INT AUTO_INCREMENT PRIMARY
KEY, name varchar(20), dept varchar(10), salary int(10))");
 System.out.println("Table Created");

 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 }
}
```

```
//Inserting data into Database using JDBC
import java.sql.*;
public class InsertTable
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee"
,"root","");
Statement st=con.createStatement();

 st.executeUpdate("INSERT INTO emp(name,dept,salary)
 VALUES ('Bharath','Developer',74867)");
 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 System.out.println("Row Inserted");
 }
}
```

```
//Retrieve data from Database using JDBC
import java.sql.*;
public class GetTable
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee"
,"root","");
 Statement st=con.createStatement();
 ResultSet rs = st.executeQuery("select * from emp");

 System.out.println("The Employee Details are \n");
 while(rs.next())
 {
 System.out.println(rs.getInt(1)+":"+rs.getString(2)+":"+
 rs.getString(3)+":"+rs.getInt("salary"));
 System.out.println();
 //rs.getInt("id");
 //rs.getString("name");
 }

 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 }
}
```

```
//Update data in Database using JDBC
import java.sql.*;
public class UpdateTable
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee"
,"root","");
 Statement st=con.createStatement();

 st.executeUpdate("UPDATE emp SET dept='Designer' WHERE id=101");
 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }

 System.out.println("Row Effected");
 }
}
```

```
//Delete data in Database using JDBC

import java.sql.*;
public class DeleteTable
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee"
,"root","");
 Statement st=con.createStatement();
 st.executeUpdate("DELETE FROM emp WHERE id=121");
 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 System.out.println("Row Effected");
 }
}
```

```
//ResultSetMetadata class
import java.sql.*;
public class MetaTable
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee"
,"root","");
 Statement st=con.createStatement();
 ResultSet rs = st.executeQuery("select * from emp");
 ResultSetMetaData a = rs.getMetaData();

 System.out.println("The column count is " +
a.getColumnCount());
 System.out.println("The column count is " +
a.getTableName());

 for(int i=1 ; i<=4 ;i++)
 {
 System.out.println("Column "+i+ a.getColumnName(i));
 System.out.println("Column Type "+i+
a.getColumnTypeName(i));
 }

 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 }
}
```

```
//Insert records using PreparedStatement
import java.sql.*;
import java.util.*;

public class PrepareTable
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection con=DriverManager.getConnection
 ("jdbc:mysql://localhost:3306/employee","root","");
 PreparedStatement st=con.prepareStatement("INSERT INTO
emp VALUES(?,?,?,?,?)");

 Scanner sc = new Scanner(System.in);
 System.out.println("Enter Emp Id");
 int id1= sc.nextInt();
 System.out.println("Enter Emp Name");
 String na = sc.next();
 System.out.println("Enter Emp Dept");
 String dp = sc.next();
 System.out.println("Enter Emp Salary");
 int sa = sc.nextInt();

 st.setInt(1,id1);
 st.setString(2,na);
 st.setString(3,dp);
 st.setInt(4,sa);

 int i=st.executeUpdate();
 System.out.println(i+" records inserted");
 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 System.out.println("Row Inserted");
 }
}
```

```
//retrieve records based on parameter using PreparedStatement
import java.sql.*;
import java.util.*;
public class PrepareTable1
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee"
,"root","");
 PreparedStatement st=con.prepareStatement("SELECT *
from emp where dept=?");

 Scanner sc = new Scanner(System.in);
 System.out.println("Enter Department Name");
 String dpt= sc.next();

 st.setString(1,dpt);
 ResultSet rs = st.executeQuery();
 System.out.println("The Employee Details are \n");
 while(rs.next())
 {
 System.out.println(rs.getInt(1)+":"+rs.getString(2)+":"+
rs.getString(3)+":"+rs.getInt("salary"));
 System.out.println();
 }
 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 System.out.println("Row Inserted");
 }
}
```

```
//CallableStatement Demo - insertion - with Procedure CallSite
//goto database - select routines - add routine - insert arguments - write
query - submit.
import java.sql.*;
import java.util.*;
public class CallableDemo
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee"
,"root","");
 CallableStatement st=con.prepareCall("{call INSERTINPUT(?,?,?,?,?)}");
 Scanner sc = new Scanner(System.in);
 System.out.println("Enter Emp Id");
 int id1= sc.nextInt();
 System.out.println("Enter Emp Name");
 String na = sc.next();
 System.out.println("Enter Emp Dept");
 String dp = sc.next();
 System.out.println("Enter Emp Salary");
 int sa = sc.nextInt();

 st.setInt(1,id1);
 st.setString(2,na);
 st.setString(3,dp);
 st.setInt(4,sa);

 st.execute();
 con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 }

 System.out.println("Row Inserted");
}
}
```

ODBC	JDBC
1) ODBC Stands for Open Database Connectivity	1) JDBC Stands for Java Database Connectivity
2) Introduced by Microsoft.	2) Introduced by Sun Micro Systems.
3) We can Use ODBC for any Languages like C, C++, Java, Etc.	3) We can Use JDBC only for Java Language.
4) We can use ODBC only for Windows Platforms.	4) We can use JDBC for any Platform.
5) Mostly ODBC Drivers are developed in Native Languages like C OR C++.	5) Mostly JDBC Drivers are developed in Java.
6) For Java Applications, it is not recommended to use ODBC because Performance will be Down due to Internal Conversions and Application will become Platform Dependent.	6) For Java Applications, it is highly recommended to use JDBC because there is no Performance Problems and Platform Dependency Problems.

executeQuery()	executeUpdate()	execute()
This method is used to execute the SQL statements which retrieve some data from the database.	This method is used to execute the SQL statements which update or modify the database.	This method can be used for any kind of SQL statements.
This method returns a ResultSet object which contains the results returned by the query.	This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing.	This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or returned nothing.
This method is used to execute only select queries.	This method is used to execute only non-select queries.	This method can be used for both select and non-select queries.
Ex : SELECT	Ex : DML → INSERT, UPDATE and DELETE DDL → CREATE, ALTER	This method can be used for any type of SQL statements.

```
//ResultSetMetadata class
import java.sql.*;
public class MetaTable
{
 public static void main(String args[])
 {
 try
 {
 Class.forName("com.mysql.jdbc.Driver");
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee"
,"root","");
 Statement st=con.createStatement();
 ResultSet rs = st.executeQuery("select * from emp");
 ResultSetMetaData a = rs.getMetaData();

 System.out.println("The column count is " + a.getColumnCount());
 System.out.println("The table name is " + a.getTableName());

 for(int i=1 ; i<=4 ;i++)
 {
 System.out.println("Column "+i+ a.getColumnName(i));
 System.out.println("Column Type "+i+ a.getColumnTypeName(i));
 }

 con.close();

 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 }
}
```