

# Web Programming Technologies

## Introduction to WEB Session-01

Harshita Maheshwari

# Agenda for Today's Session

- Introduction to Web Programming
- What is Internet?
- Uses of Internet
- How the Internet works?
- DNS
- What is Web?
- How web works?
- Web Servers
- HTTP Protocol
- HTTP Connections
- Stateless nature of HTTP
- HTTP Request and Response Message
- Request Methods
- Status codes
- HTTPS



# Introduction

In the current information age, the Web plays an important role in connecting people, sharing relevant content and enabling us to conduct our daily activities.

# So It Begins

**Principals of web application, web application workflow, Languages for web programming, Database-driven websites, authentication and session, testing and debugging.**

# When it ends.....

You should;

- Have an understanding of the importance of web technologies
- Be able to design and build websites
- Be able to build static and dynamic WebPages
- Integrate websites with databases
- Develop basic applications using a variety of internet programming languages
  
- But in reality it can be much, much more.....

# Where it is Heading

- Web Designer
- Web Developer
- Full Stack Web Developer
- UI Developer
- Front End Developer

# What is web programming

Web programming is used to build:

- **Web Pages:**

Web page is an electronic document written in a computer language called HTML. These web pages are linked together through a system of connections called hypertext links, which enable the user to jump from one web page to another by clicking on a link.

- **Websites:**

A collection of related web pages found at a single address.

A URL serves as the top-level address of a Web site and points to that Website's home page. That page serves as a reference point, containing pointers to additional HTML pages or links to other websites.

## **Types of Websites : Static Website and Dynamic Website**

## ➤ Web Applications:

A web App is a set of WebPages that are generated in response to user requests

Ex. Search engines, online stores, auctions, news sites, games etc.

# Static Website

**Static websites contain fixed number of pages and format of web page is fixed which delivers information to the client.**

- **Time saving:** A static website is quick to develop.
- **Cost Effective:** Static websites are cheaper to develop.
- **Fast transferring:** Static websites do not have complex structures and can be easily and quickly transferred from server to client without much processing time.
- **Difficult to change:** All the HTML files would need to be individually changed even for a small change made to the website which takes lot of time.
- **Limited Functionality:** It does not offer all the functionalities that a dynamic website can.

# Dynamic Website

**Dynamic websites can change the web page contents dynamically while the page is running on client's browser.**

- **Easy to update:** No expert knowledge is needed in changing the dynamic website.
- **Interactive:** Dynamic websites interact with the users and changes according to their behavior.
- **Quick to Responsiveness:** Can be quickly updated to become responsive to various screen sizes.
- **Smooth Navigation:** User can jump from one page to the other without any problem.

# What is Internet??

- A computer network made up of thousands of networks worldwide
- It consists of millions of computers interconnected with one another via a modem and Transmission Control Protocol/Internet Protocol suite, (TCP/IP)
- Internet is a global network of computers which are mostly connected using telephone lines/satellites for the purpose of sharing information



# Uses of Internet



## USES OF THE INTERNET

ON AVERAGE,  
**139,344**



GO LIVE  
each day

USES OF THE INTERNET IN  
*percentage of users*



**62%**  
RESEARCH



**50.1%**  
BANKING



**58%**  
SHOPPING



**15.2%**  
MEETING PEOPLE



**62.2%**  
INFORMATION  
ABOUT HEALTH



**43%**  
MAKING TRAVEL  
RESERVATIONS



**45.5%**  
LOOKING  
FOR JOBS

# Internet Service Provider

Is a company that provides connection to the internet.

Jio

Airtel

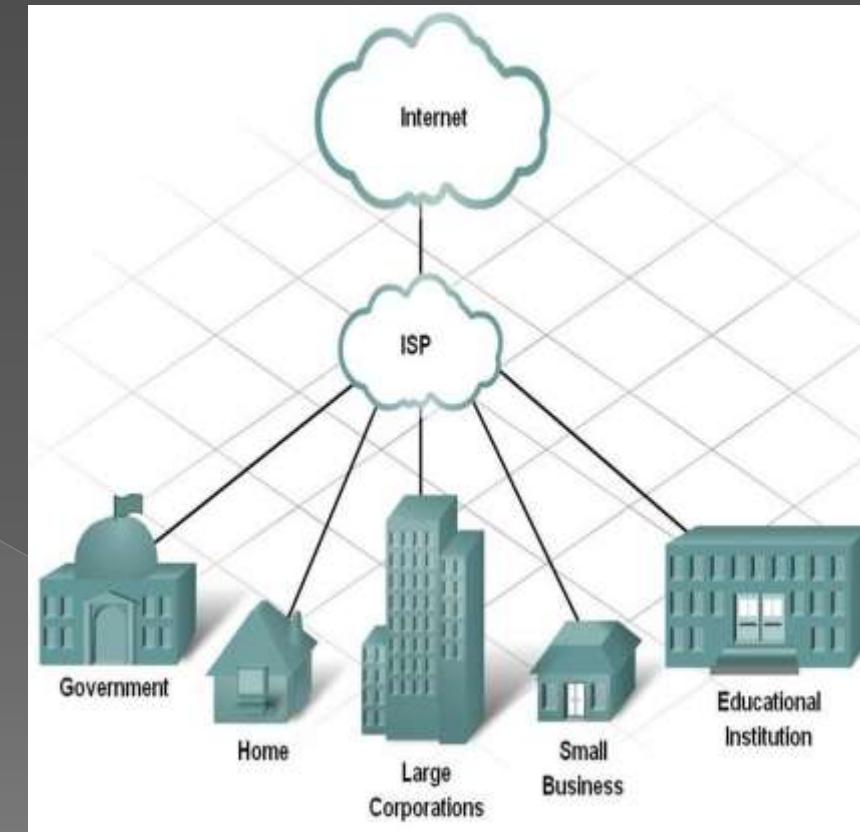
Vodafone Idea

BSNL

MTNL

Hathway

ACT Fibernet

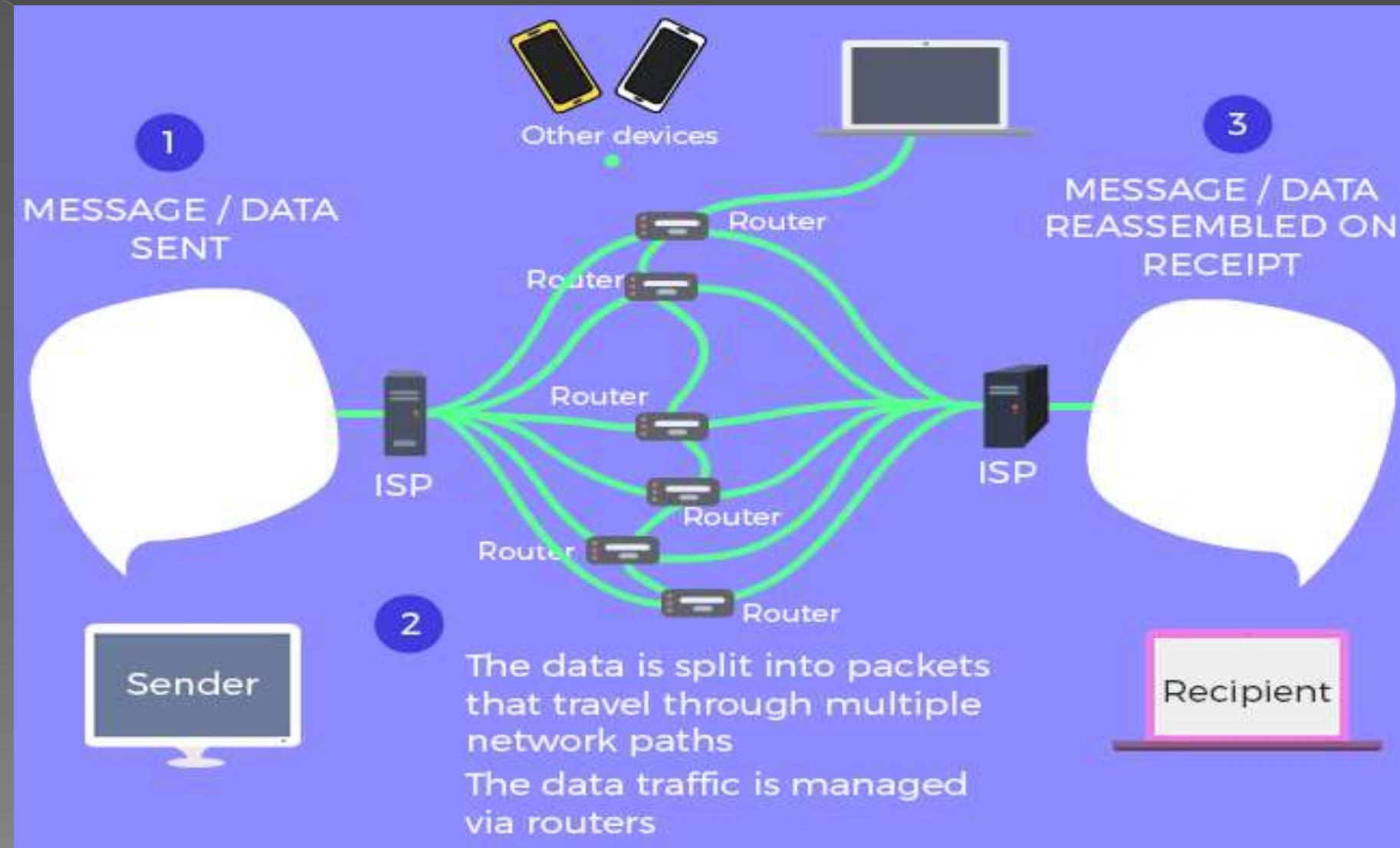


# Web Browser



**Is a program that is used to navigate the internet.**

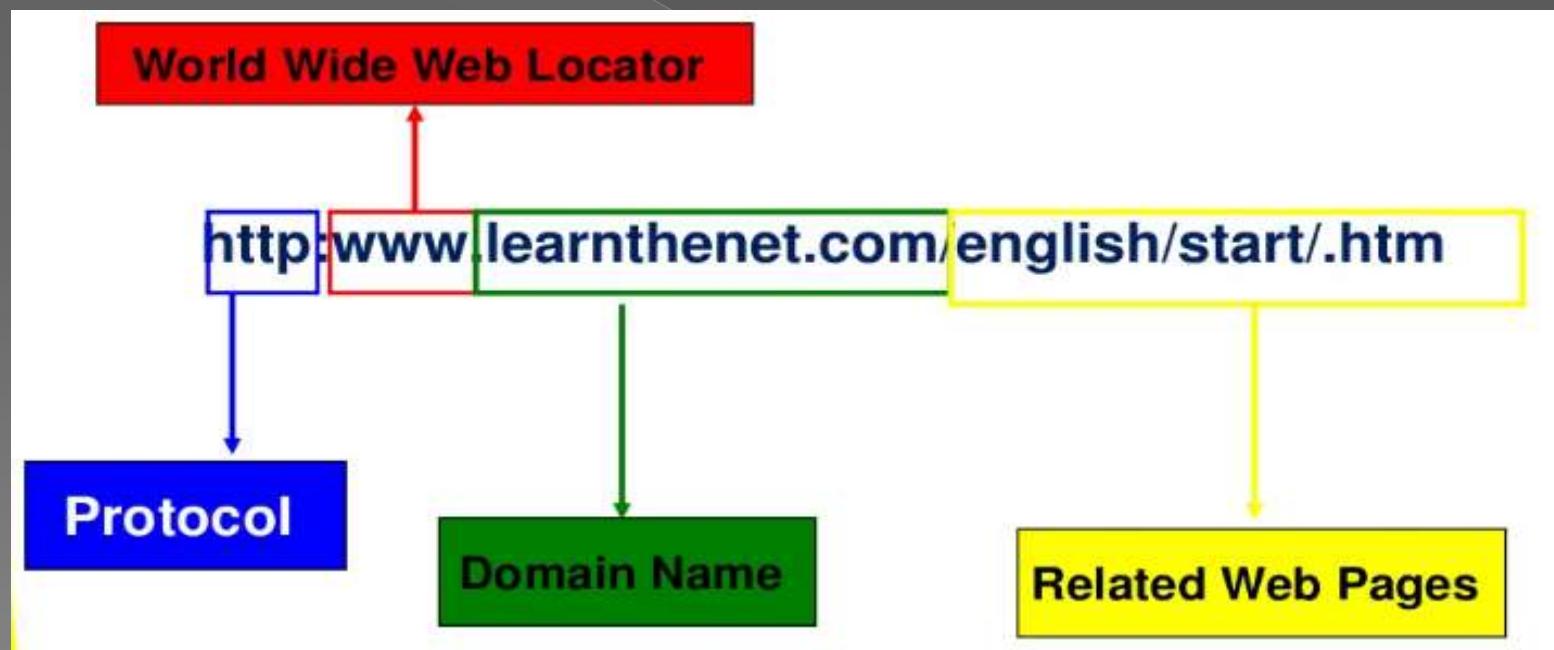
# How the Internet Works??



# URL

- URL is the address of an object, a document, a file or a web page on the internet.

## Parts of URL:



## **http:/.(Hypertext Transfer Protocol)**

- Protocol used in the communication between the browser and the web server.

## **WWW:**

- The Web consists of pages that can be accessed using a Web browser.

# Domain Name System

- Tells the name of the owner of the site.
- Suffix type extension name that tells what the site is about.
- Sometimes it has country code that tells where the website is located.
- Learnthenet- name of the owner
- .com- a suffix name that tells a site is a commercial site.



## Related web page:

- English/start.htm
- Is a directory or folder connected to the web that contains a group of related web pages.

# WWW or Web or W3??

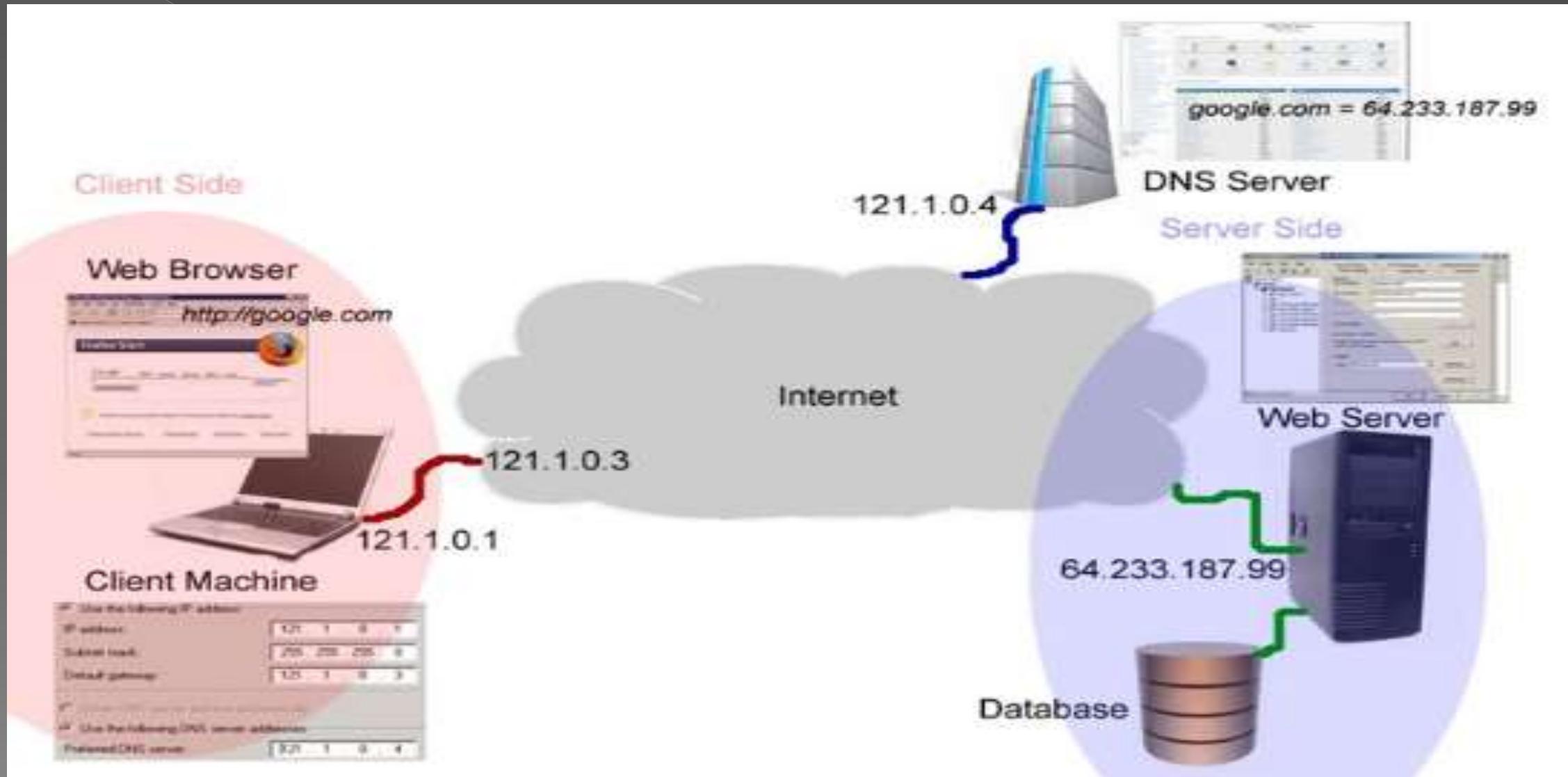
- The World Wide Web is a way of exchanging information between computers on the Internet.
- The World Wide Web is the network of pages of images, texts and sounds on the Internet which can be viewed using browser software .
- The World Wide Web, or Web, consists of a worldwide collection of electronic documents (Web pages).
- 1989 – 1990 Tim Berners Lee invents the World Wide Web (www)



# Difference between Web and Internet

- Many people use the terms Internet and World Wide Web, but in fact the two terms are not synonymous. The Internet and the Web are two separate but related things.
- The Internet is a massive network of networks. It connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are both connected to the Internet.
- The World Wide Web, or simply Web, is a way of accessing information over the medium of the Internet.
- So the Web is just a portion of the Internet, albeit a large portion, but the two terms are not synonymous and should not be confused.

# How Web Works?



# Web Server

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

- **IIS Server** :- runs on the Microsoft .NET platform on the Windows OS
- **Apache Server** :- most widely used web server software.  
Developed and maintained by Apache Software Foundation, Apache is an open source software available for free. It runs on 67% of all web servers in the world. It is fast, reliable, and secure. It can be highly customized to meet the needs of many different environments by using extensions and modules.

# HTTP

# Protocol:

A Protocol is a standard procedure for defining and regulating communication. For example TCP, UDP, HTTP etc.

- HTTP is the foundation of data communication for the World Wide Web.
- HTTP is the protocol that web browsers and web servers use to communicate.
- The HTTP is the Web's application- layer protocol for transferring various forms of data between server and client like plaintext, hypertext, image, videos and sounds .



# HTTP Connections

## Non-persistent connection:

- Each request/response pair are sent over a separate TCP connection.
- HTTP1.0

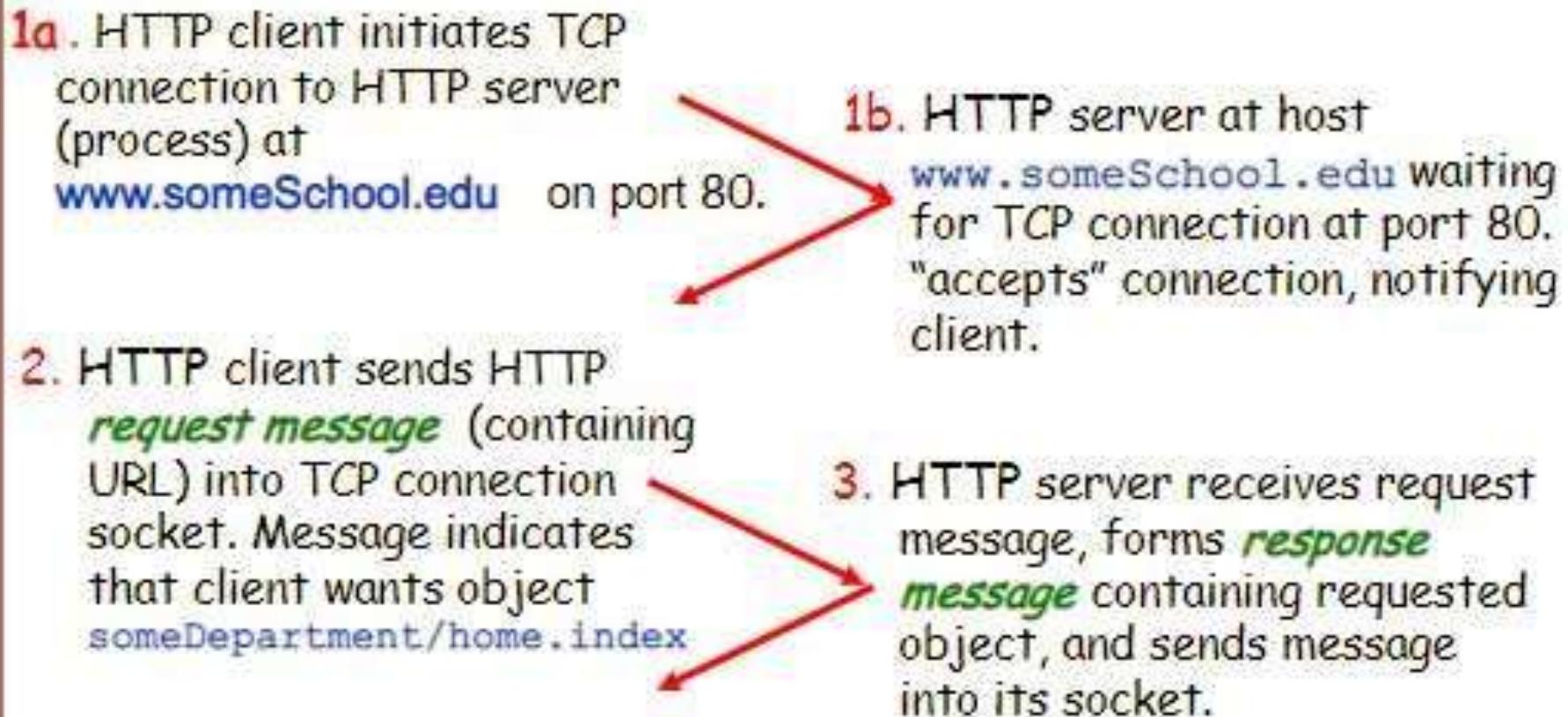
## Persistent connection:

- All of the requests and their corresponding responses are sent over the same TCP connection.
- HTTP1.1

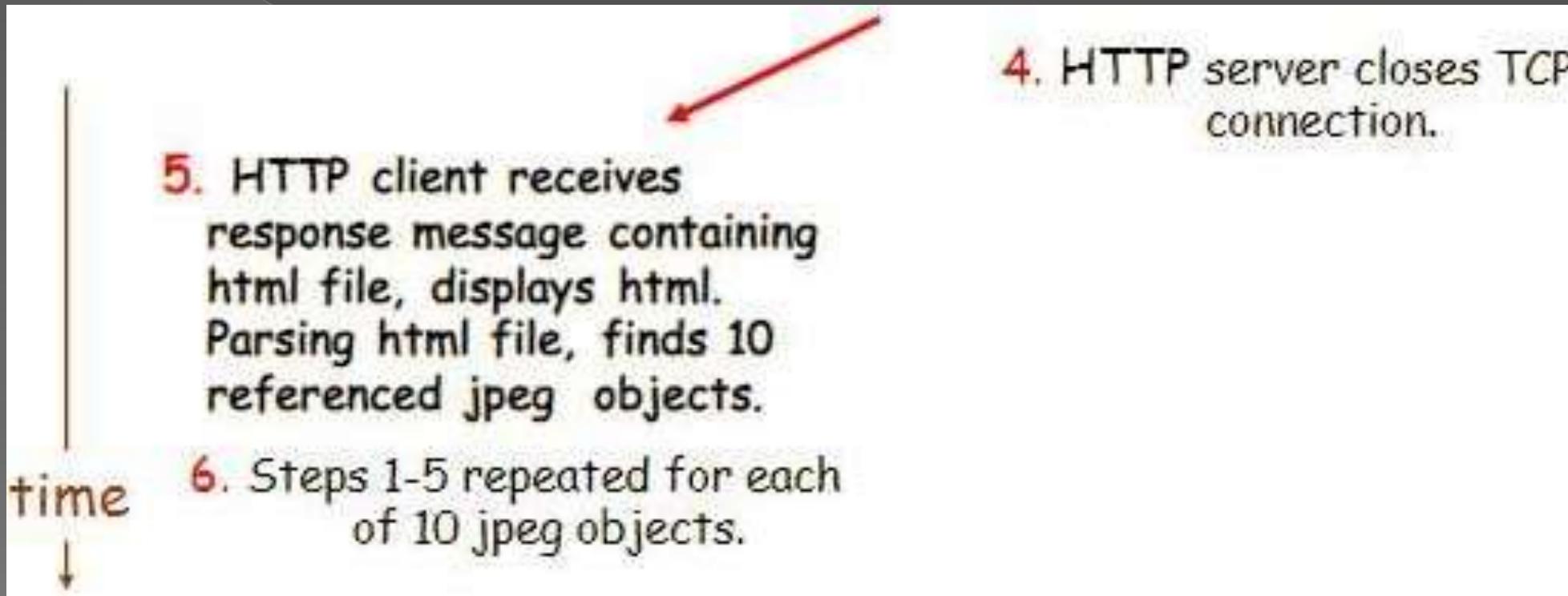
# Nonpersistent HTTP

Suppose user enters URL

`www.someSchool.edu/someDepartment/home.index`

- 
- 1a. HTTP client initiates TCP connection to HTTP server (process) at `www.someSchool.edu` on port 80.
  - 1b. HTTP server at host `www.someSchool.edu` waiting for TCP connection at port 80. "accepts" connection, notifying client.
  2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `someDepartment/home.index`
  3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket.

time  
↓



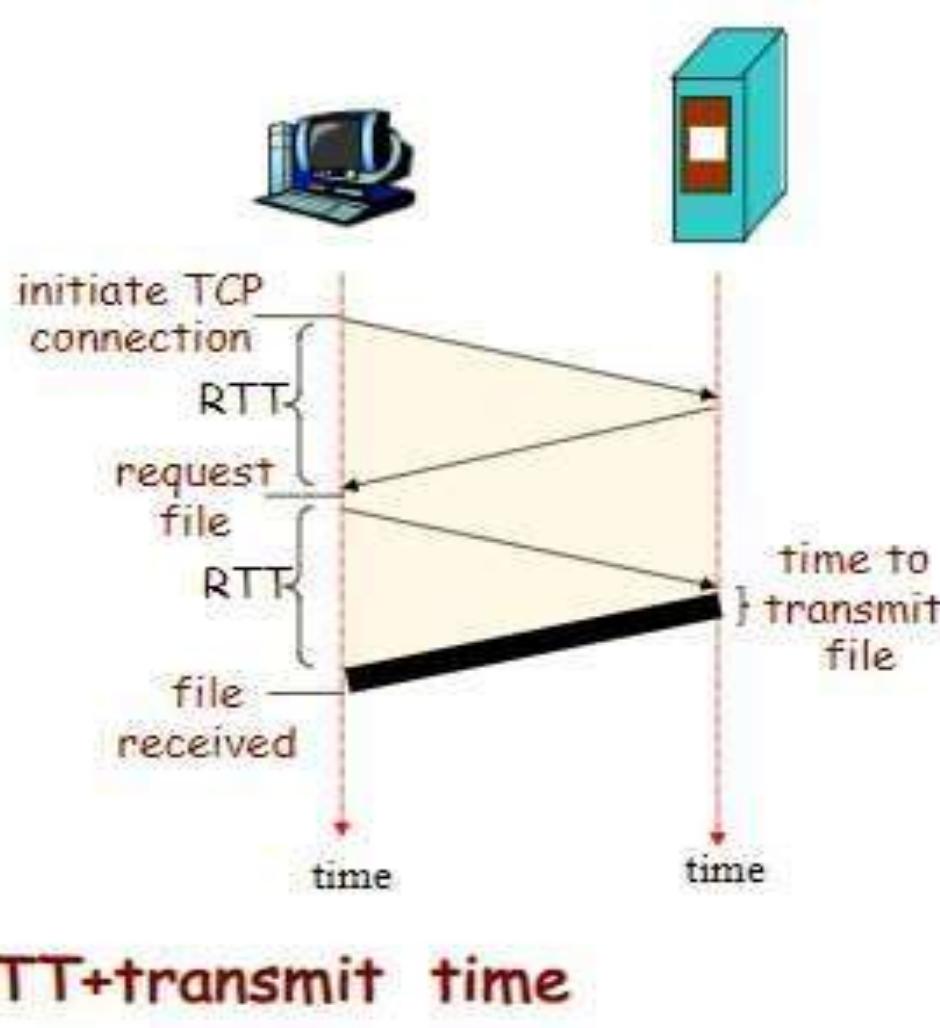
# Nonpersistent HTTP: Response Time

**Definition of RTT:** time for a small packet to travel from client to server and back.

**Response time:**

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

$$\text{total time} = \text{2RTT} + \text{transmit time}$$



# Persistent HTTP

## Nonpersistent HTTP issues:

- requires 2 RTTs per object.
- OS overhead for *each* TCP connection.
- browsers often open parallel TCP connections to fetch referenced objects.

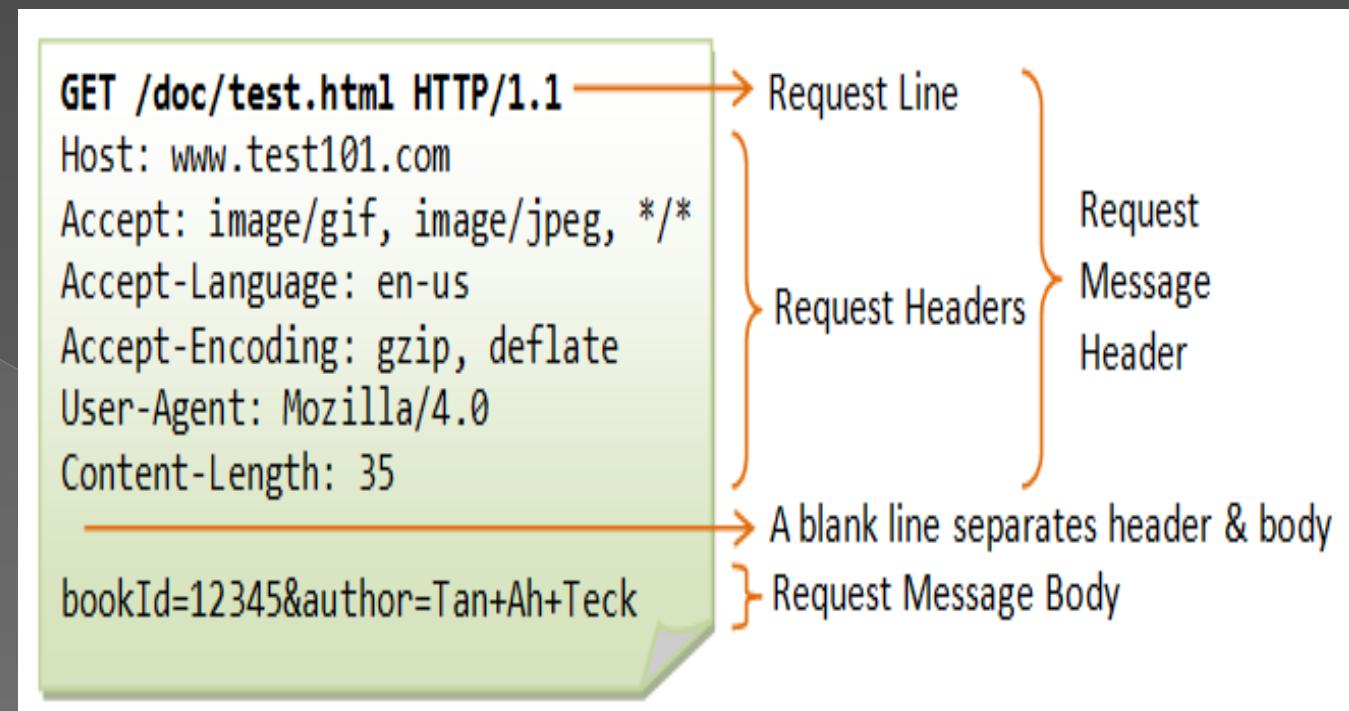
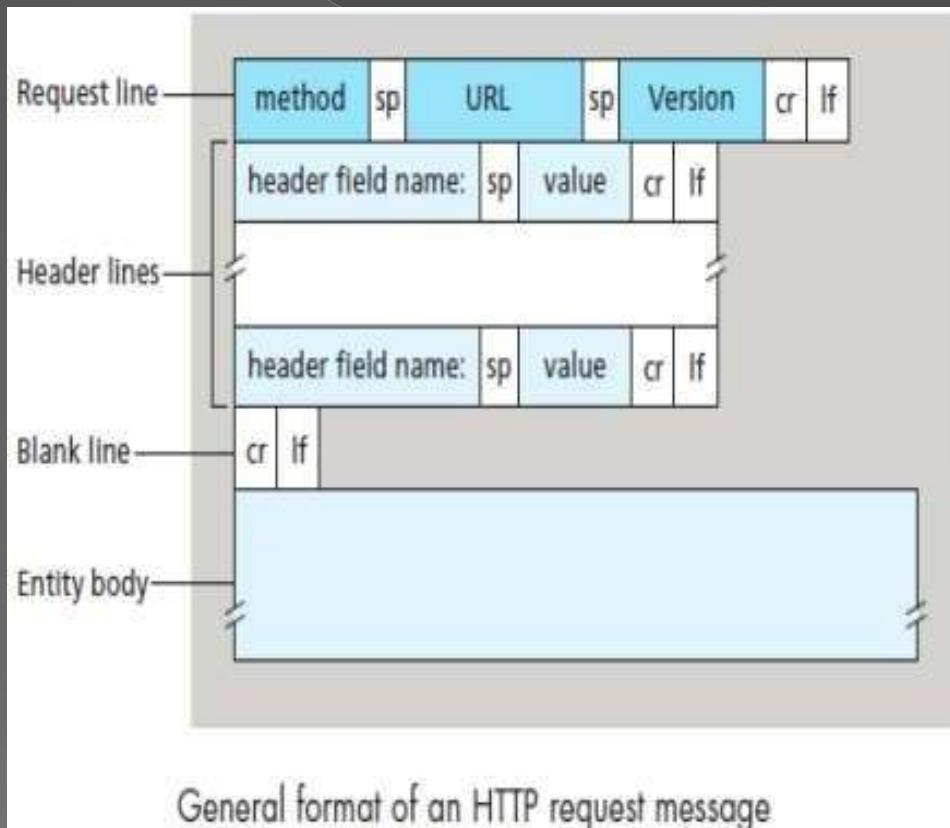
## Persistent HTTP

- server leaves connection open after sending response.
- subsequent HTTP messages between same client/server sent over open connection.
- client sends requests as soon as it encounters a referenced object.
- as little as one RTT for all the referenced objects.

# Stateless Nature of Http

- HTTP is a stateless protocol, because an HTTP server maintains no information about the clients.
- If a particular client asks for the same object twice in a period of a few seconds, the server does not respond by saying that it just served the object to the client; instead, the server resends the object, as it has completely forgotten what it did earlier.

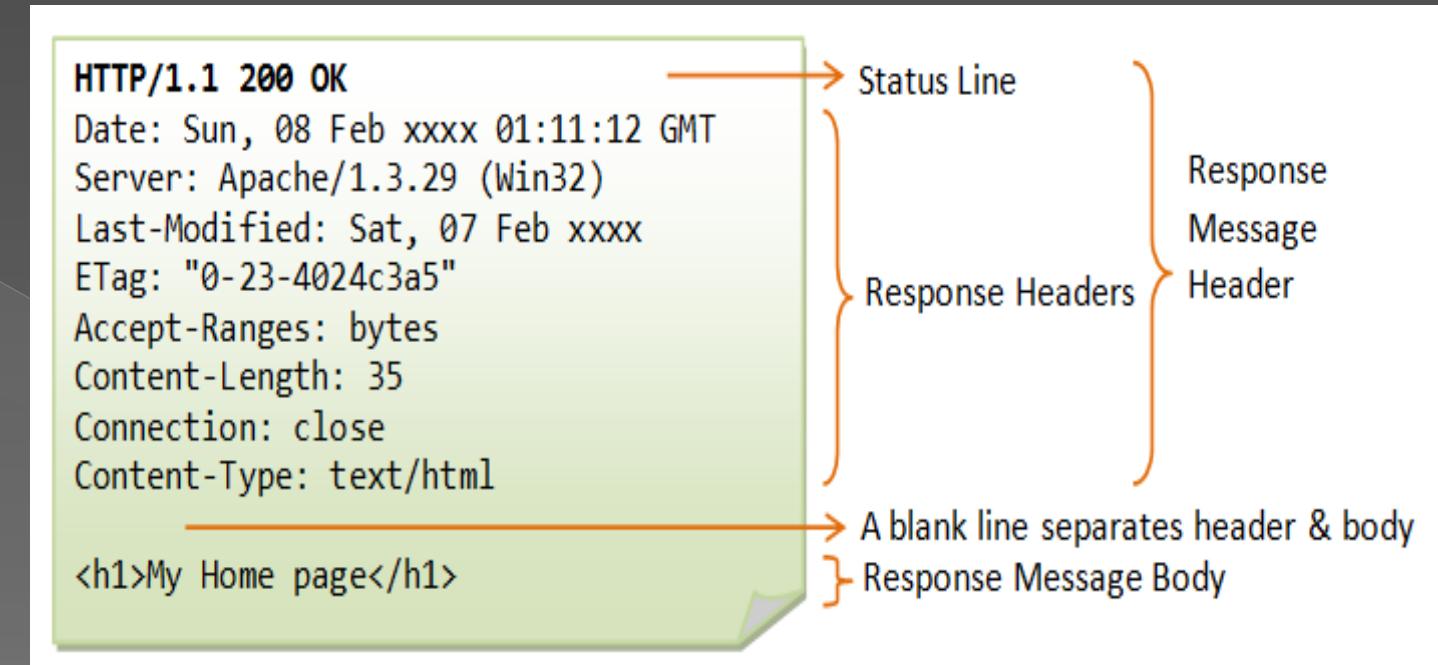
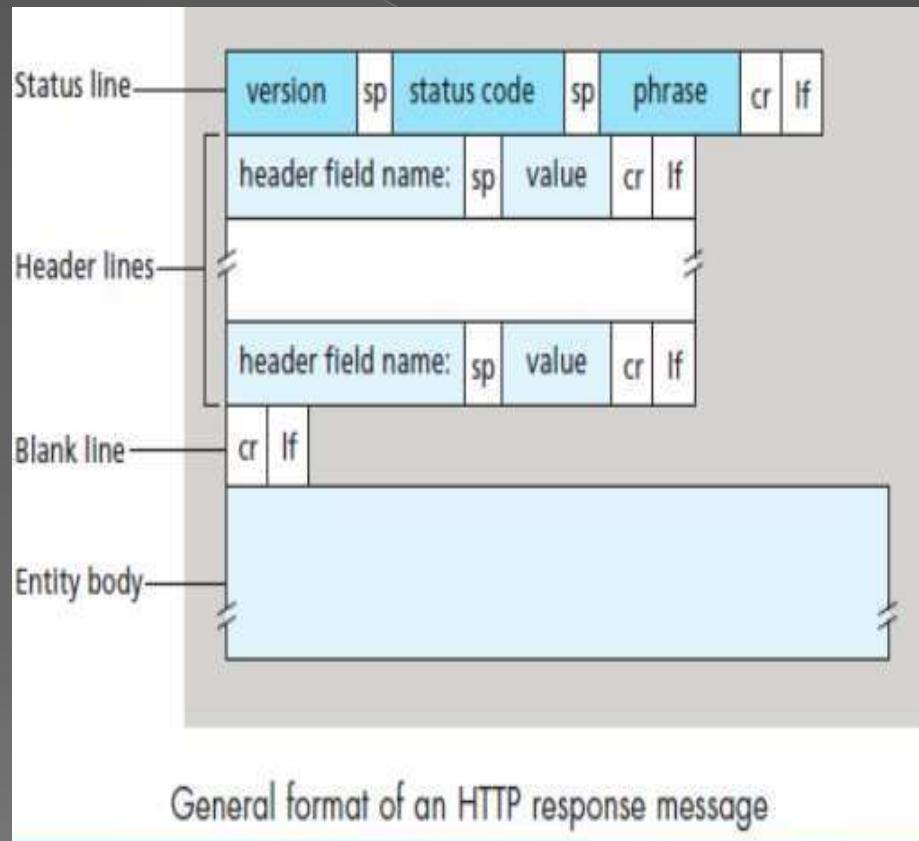
# Http Request Message



# Request Methods

Method	Description
GET	Request for resource from server
POST	Submit data to the server
HEAD	Same as GET but does not return the body
PUT	The data within the request must be stored at the URL supplied, replacing any existing data.
DELETE	Delete a resource
OPTIONS	Return the HTTP methods supported by the server
CONNECT	Client requests the HTTP proxy to forward a TCP connection to some destination. Used to create a TCP/IP tunnel for secure connections using HTTP proxies.

# Http Response Messages

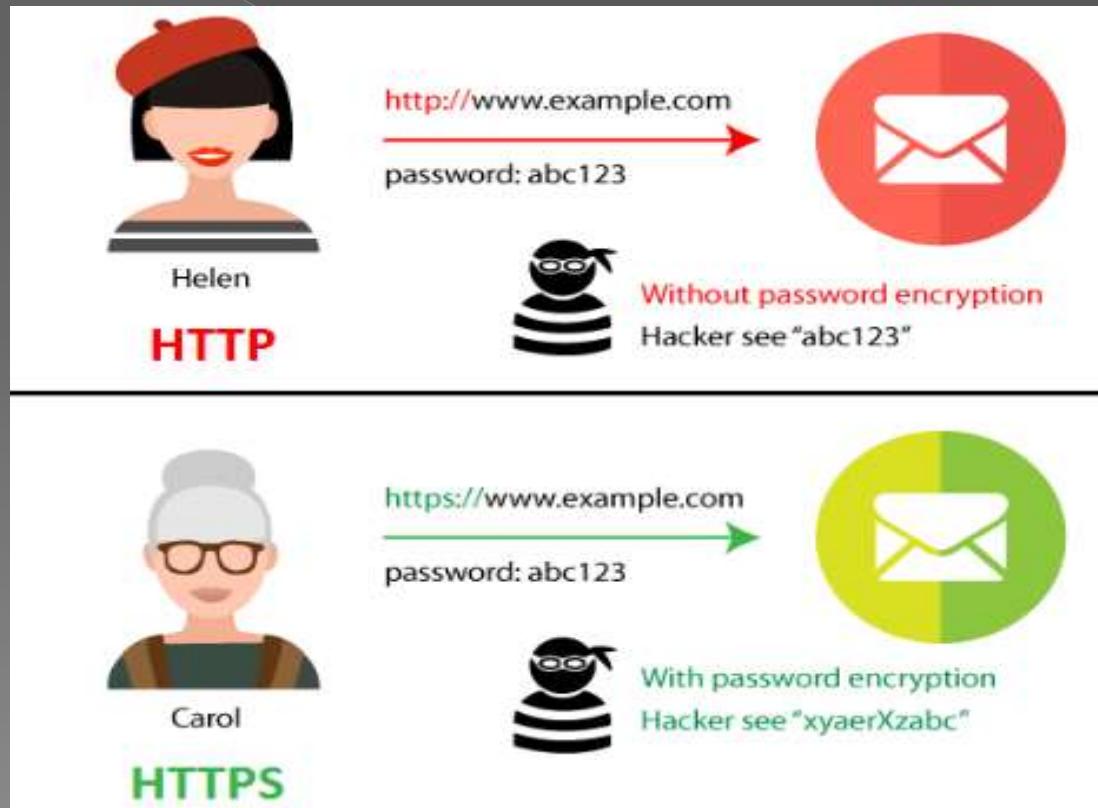


# Status Codes

Code	Description	Code	Description
200	OK	400	Bad Request
201	Created	401	Unauthorized
202	Accepted	403	Forbidden
301	Moved Permanently	404	Not Found
303	See Other	410	Gone
304	Not Modified	500	Internal Server Error
307	Temporary Redirect	503	Service Unavailable

# HTTPS

Hypertext transfer protocol secure (HTTPS) is the secure version of HTTP, which is the primary protocol used to send data between a web browser and a website. HTTPS is encrypted in order to increase security of data transfer. This is particularly important when users transmit sensitive data, such as by logging into a bank account, email service, or health insurance provider.



# Quiz

**Q1. In the process of fetching a web page from a server the HTTP request/response takes \_\_\_\_\_ RTTs.**

- A. 2
- B. 1
- C. 4
- D. 3

**Answer: B**

1

**Q2. The values GET, POST, HEAD etc are specified in \_\_\_\_\_ of HTTP message**

- A. Request line
- B. Header line
- C. Status line
- D. Entity body

**Answer: A**

**Request Line**

**Q3** HTTP is \_\_\_\_\_ protocol.

- A. application layer
- B. transport layer
- C. network layer
- D. none of the mentioned

**Answer:** A

**Application Layer Protocol**

**Q4. Multiple object can be sent over a TCP connection between client and server in**

- A. persistent HTTP
- B. nonpersistent HTTP
- C. both (a) and (b)
- D. none of the mentioned

**Answer: A  
persistent HTTP**

## **Q5. URL stands for**

- A. unique reference label**
- B. uniform reference label**
- C. uniform resource locator**
- D. unique resource locator**

**Answer: C**  
**uniform resource locator**

# Web Programming Technologies

## HTML - Basics Session-02

Harshita Maheshwari

# Agenda for Today's Session

- What is HTML?
- Why HTML is used?
- HTML Page Structure
- HTML Element & Attribute
- HTML Tag & Types of Tag
- Basic HTML Tags
- Heading, paragraph, div, span, br, hr, pre etc.
- Text Formatting Tags
- List Tags – ol, ul and definition list
- Table
- Img tag
- Anchor tag
- HTML Entities
- Marquee Tag
- FieldSet Tag



# Introduction

- **HTML is a language for describing web pages.**
- **HTML stands for Hyper Text Markup Language**
- **HTML is not a programming language, it is a markup language**

**HYPertext** means ordinary text that has extra features of linking.

**MARKUP** means process of adding display related features.

**Language** has its own syntax and rules.

# **Three Layer of Web Design:**

## **Structure, style and Behavior**

## BEHAVIOR

Javascript



## PRESENTATION

CSS

Imagery



## STRUCTURE

HTML markup

Site planning



# Why HTML is Used?

- **Easy and Simple language**
- **Markup language:** It provides a flexible way to design web pages along with the text.
- **Platform Independent :** It can display on any platform like Window, Linux etc.
- **Attractive and interactive:** It facilitates the programmers to add Graphics, Videos and sound to the web pages.
- **Effective Presentation :** It has a lot of formatting tags for effective presentation
- **Support to Scripting language:** It support scripting languages to create dynamic web applications.



# HTML Page Structure

**HTML web page is divided into three parts:**

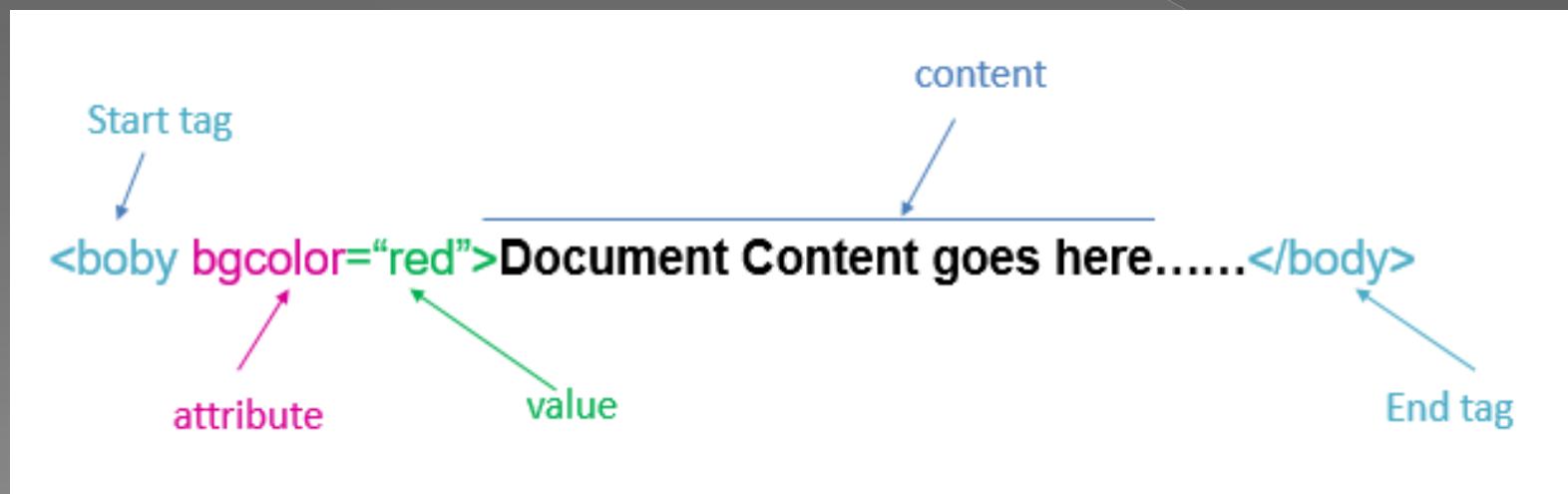
- **Comment Section (optional)** : This section contains comments about the web page.
- **Head Section (optional)** : The head section is defined with a starting <head> tag and closing </head> tag. This section usually contains a title for the web page.
- **Body section** : The body section comes after the head section. The body section contains the entire information about the web page and its behavior.

**Example:**

```
<!DOCTYPE html>
<!-- First HTML Program -->
<html>
<head><title> Introduction to HTML </title></head>
<body>
  HTML stands for Hypertext Markup Language
</body>
</html>
```

# HTML Element

- An HTML element is the collection of start tag, its attributes, an end tag and everything in between.
- HTML elements are the building blocks of HTML pages.
- HTML uses elements to specify a document's structure, to provide information and to format its contents.
- HTML elements are defined or represented by using HTML tags.



# Attribute

- Attributes add discretionary properties to tags
- They are added to the opening tag within the <> brackets.  
For example  
`<html lang="en">`
- adds a language attribute to the `<html>` tag which is useful for search engines and assistive devices
- Attributes can use single or double quotes – or no quotes

# HTML Tag

**Tags are case-insensitive and predefined.**

HTML documents are simply a text made up of HTML elements and these elements are defined using HTML tags.

Tags contain elements which provide instructions for how information will be processed or displayed on a web page.

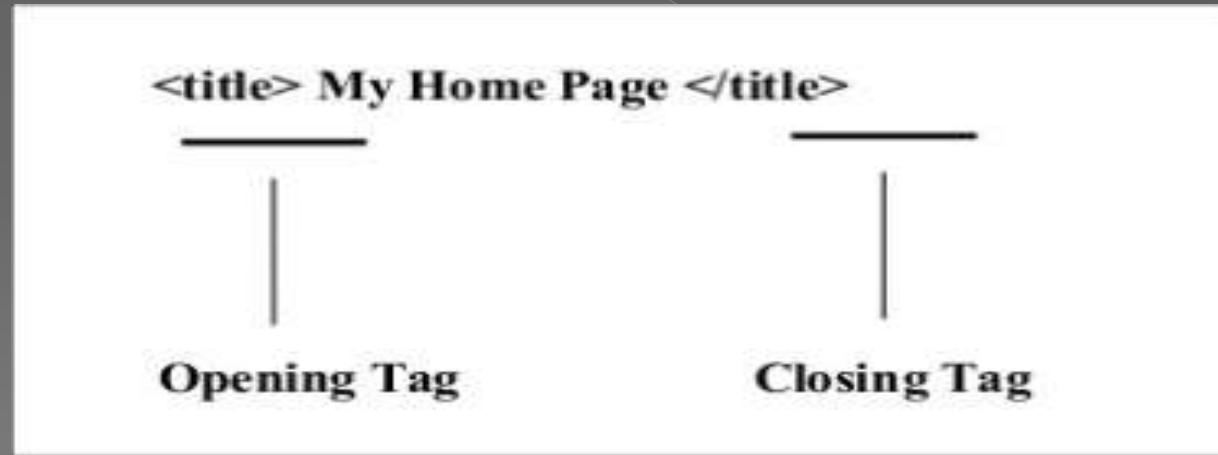
The tag is an HTML command that shows the layout or displays the desired output of a whole or part of the web page.

## **Types of Tag:**

- Container Tag
- Empty Tag

# Container Tag

- Container tag contain text between an opening and a closing tag.
- Also known as non-empty tags.



# Empty Tag

An empty tag does not use a closing tag.

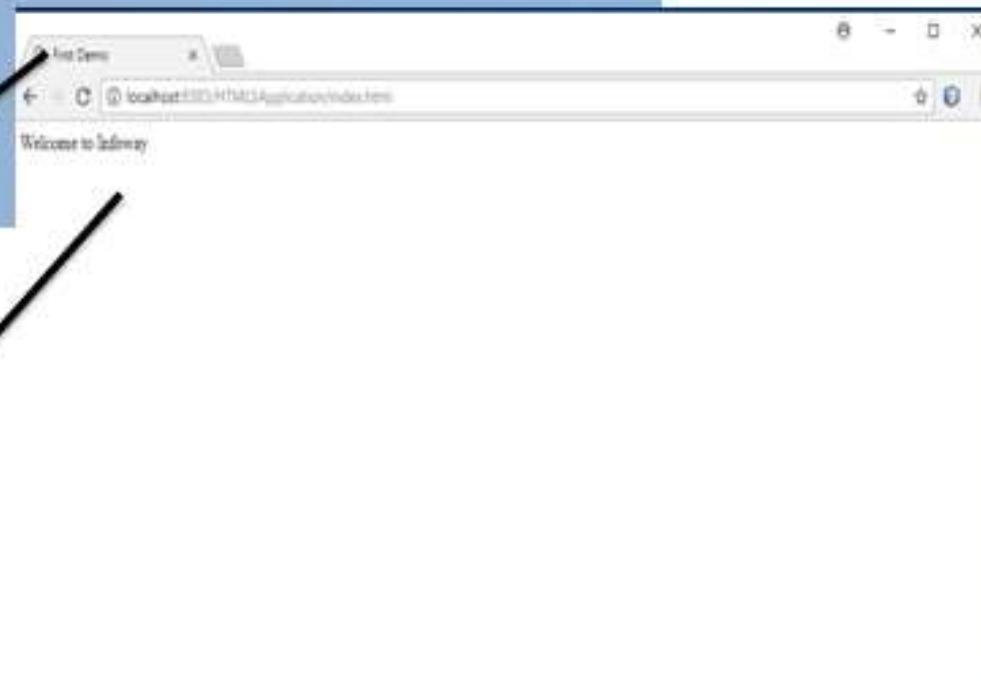
<BR> Break a line  
<BR> Break a line



Empty Tags

# First HTML Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>First Demo</title>
    <meta charset='UTF-8'>
    <meta name='viewport' content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <div>Welcome to Infoway</div>
  </body>
</html>
```



# HTML Tag

The `<html> </html>` tag tells the browser that this is an HTML document.

The `html` element is the outermost element in HTML.

# <!DOCTYPE> Declaration

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

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
  <meta charset="utf-8" />
</head>
<body>
</body>
</html>
```

# Head Tag

The `<head></head>` element is a container for metadata (data about data) and is placed between the `<html>` tag and the `<body>` tag.

`<head>` element can include :

- `<title></title>`
- `<style></style>`
- `<link/>`
- `<script></script>`
- `<meta> </meta>`tags and more

# Title Tag

**<title> will be shown on the browser toolbar and also provides a title for the page when it is added to favorites.**

```
<!DOCTYPE html>
<html>
  <head>
    <title> Infoway Technologies, PUNE</title>
  </head>
</html>
```

# Meta Tag

- Metadata is data (information) about data.
- Give a short description about the document. This can be used by various search engines while indexing your webpage for searching purpose.
- <meta> are typically used to specify page description, keywords, author of the document, last modified, and other metadata.
- Metadata will not be displayed on the page, but will be machine parsable.

Attribute	Value	Description
charset	character_set	Specifies the character encoding for the HTML document
content	text	Gives the value associated with the http-equiv or name attribute
http-equiv	refresh	Will refresh the document after specified seconds
name	application-name author description keywords viewport	Specifies a name for the metadata

```
<head>
  <meta charset="UTF-8">
</head>
```

```
<head>
  <meta name="author" content="Author Name">
  <meta name="description" content="HTML Tutorials">
  <meta name="generator" content="Netbeans">
  <meta name="keywords" content="HTML, meta tag, tag
    reference">
  <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
</head>
```

```
<head>
  <meta http-equiv="refresh" content="30">
</head>
```

# Body Tag

**The body element defines the document's body.**

**The <body> tag creates the body section of the document, which contains the actual visible content of the document.**

**Attributes:-**

**bgcolor:** used to set background color.

```
<body bgcolor="color"></body>
```

**text:** Used to control the color of all the normal text in the document. The default color for text is black.

```
<body text="color"></body>
```

# Heading Tag

**<h1>** to **<h6>** tags are used to define HTML headings.

**<h1>** defines the most important heading.

**<h6>** defines the least important heading.

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

**Font Size:-**

**h1 is 32px**

**h2 is 24px**

**h3 is 20.8px**

**h4 is 16px**

**h5 is 12.8px**

**h6 is 11.2px**

# Paragraph Tag

The < p > tag defines a paragraph.

The p element automatically creates some space before and after itself. The space is automatically applied by the browser, or you can specify it in a style sheet.

**< p > Paragraph text... < /p >**

Attribute	Description
Align	Left/right/center/justify
contenteditable	Specifies whether the content of an element is editable or not
tabindex	Specifies the tabbing order of an element

## HTML Code

```
<P ALIGN="left"> This paragraph is  
left aligned. </P>
```

```
<P ALIGN="CENTER"> This is a  
centered paragraph. </P>
```

```
<P ALIGN="RIGHT"> This paragraph  
is right aligned. </P>
```

## Browser Display

This paragraph is left aligned.

This is a centered paragraph.

This paragraph is right aligned.

# **br and hr tag**

**<br > tag inserts a single line break.**

**<hr > tag creates a horizontal line in an HTML page. Used to separate content in an HTML page.**

## **Attribute**

align

size

width

noshade

## **value**

left, right, center

pixels

pixels %

noshade

## **Description**

Specifies the alignment of a <hr> element

Height of a <hr> element

Width of a <hr> element

It display in one solid color instead of a shaded color

# Preformatted text Tag

<pre> tag defines preformatted text.

Ex:

```
Hello  
Welcome  
      to Infoway!!  
</pre>
```

# Text formatting tags

- <b> tag is used to bold content. </b>
- <big> Defines big text .</big>
- <em> Define Emphasized text </em>
- <small> Defines small text </small>
- <sub> subscripted text</sub>
- <sup> superscripted text </sup>
- <strong> tag is used to bold content. </strong>
- <i> tag is used to italic content. </i>
- <u> tag is used to underline content. </u>
- <q> tag is used to quote then content. </q>

# div Tag

- The <div> tag defines a division or a section in an HTML document. It is used to group block elements to format them with CSS.

<div> tags goes here </div>

| Attribute | Description               |
|-----------|---------------------------|
| Align     | Left/right/center/justify |

# **span Tag**

**<span> tag is used to group inline -elements.**

```
<p> Hello <span> World! </span> </p>
```

# Div vs. Span

The difference between span and div is that span element is in-line and usually used for a small chunk of in-line HTML.

Whereas a div element is block-line (which is basically equivalent to having a line break before and after it) and used to group larger chunks of code.

# Font Tag

- This element is used to format the size, typeface and color of the enclosed text.
- The commonly used fonts for web pages are Arial, Comic Sans MS , Lucida Sans Unicode, Arial Black, Courier New, Times New Roman, Arial Narrow, Impact, Verdana.
- default font size in most browsers is 16px.

**Not Supported in HTML5.**

Attribute	Description
color	Specifies color of text
face	Specifies the font of text
size	Specifies the size of text

# List Tag

**<ol>** tag defines an ordered list. An ordered list can be numerical or alphabetical.

## Attributes:

**Reversed**: Specifies that the list order should be descending (9,8,7...)

**Start** : Specifies the start value of an ordered list

**Type** : 1,A,a,i,l

**Ex.**      <ol>

```
<li>Jhon</li>
<li>Sara</li>
<li>Smith</li>
```

```
</ol>
```

**<ul>** tag defines an unordered (bulleted) list.

**Attribute:**

**Type:** disc, square, circle

**Ex.**    <ul>

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

# Definition List

**<dl>** tag defines a description list. The **<dl>** tag is used in conjunction with **<dt>** (defines terms/names) and **<dd>** (describes each term/name).

**Ex.**

```
<dl>
    <dt>Term 1</dt>
    <dd>This is the definition of the first term.</dd>
    <dt>Term 2</dt>
    <dd>This is the definition of the second term.</dd>
</dl>
```

# Table Tag

**<table> tag defines an HTML table.**

Syntax: <table>....</table>

Attribute	Value	Description
align	left, right, center	Specifies alignment of a table
bgcolor	colorname	Specifies background color for a table
border	pixels	Specifies width of the borders around a table
width	pixels %	Specifies the width of a table
cellpadding	pixels	Specifies the space between the cell wall and the cell content
cellspacing	pixels	Specifies the space between cells



# th Tag

**<th> tag defines a header cell in an HTML table.**

**Syntax:** <th>.....</th>

Attribute	Value	Description
align	left, right, center, justify	Aligns the content in a cell
valign	top, middle, bottom	Specifies vertical align the contents in a cell
bgcolor	colorname	Specifies the background color of a cell.
colspan	number	Sets the number of columns a cell should span
rowspan	number	Sets the number of rows a cell should span
height	pixels %	Sets the height of a cell
width	pixels %	Sets the width of a cell

# tr Tag

**<tr> tag defines a row in an HTML table. A <tr> element contains one or more <th> or <td> elements.**

**Syntax:** <tr>.....</tr>

Attribute	Value	Description
align	left, right, center, justify	Aligns the content in a table row
valign	top, middle, bottom	Specifies vertical align the contents in a table row
bgcolor	colorname	Specifies the background color of a table row

# Caption Tag

The **<caption>** tag defines a table caption. This can either appear above or below the table.

Syntax: **<caption>.....</caption>**

Attribute	Value	Description
align	left, right, top, bottom	It defines the alignment of a caption

# td Tag

**<td> tag defines a standard cell in an HTML table. The <td> tag is used to mark up individual cells inside a table row.**

Syntax: <td>.....</td>

Attribute	Value	Description
align	left, right, center, justify	Aligns the content in a cell
valign	top, middle, bottom	Specifies vertical align the contents in a cell
bgcolor	colorname	Specifies the background color of a cell.
colspan	number	Sets the number of columns a cell should span
rowspan	number	Sets the number of rows a cell should span
height	pixels %	Sets the height of a cell
width	pixels %	Sets the width of a cell

```
<table border = "1">
<thead>
  <tr><th colspan="2">Table Header (thead)</th></tr>
</thead>
<tbody>
  <tr>
    <td rowspan="2">Cell 1 - part of tbody</td>
    <td>Cell 2 - part of tbody</td>
  </tr>
  <tr>
    <td>Cell 3 - part of tbody</td>
  </tr>
</tbody>
<tfoot>
  <tr><th colspan="2">Table Footer (tfoot)</th></tr>
</tfoot>
```

Table Header (thead)	
Cell 1 - part of tbody	Cell 2 - part of tbody
Table Footer (tfoot)	

# Img Tag

<img> tag is used to add image in web page. It is empty tag.

Syntax: .....</img>



Attribute	value	Description
alt	text	Specifies an alternate text for a image
src	URL	Specifies the URL of an image
align	top, bottom, middle, left, right	Specifies alignment of an image
border	pixel	Specifies the width of the border around an image
height	pixel	Specifies height of an image
width	pixel	Specifies width of an image
hspace	pixel	Specifies whitespace on left and right side of an image
vspace	pixel	Specifies whitespace on top and bottom of an image

# Figure and figcaption Tag

**<figure>** tag used to mark up a photo in a document.

**<figcaption>** tag defines a caption for a **<figure>** element.

```
<figure>
  
  <figcaption>Fig1. -Infoway.</figcaption>
</figure>
```

# HTML Entities

Some characters are reserved in HTML.

If you use the less than (<) or greater than (>) signs in your text, the browser might mix them with tags.

Character entities are used to display reserved characters in HTML.

Result	Description	Entity Name	Entity Number
	non-breaking space	&nbsp;	&#160;
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	double quotation mark	&quot;	&#34;
'	single quotation mark (apostrophe)	&apos;	&#39;
¢	cent	&cent;	&#162;
£	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
€	euro	&euro;	&#8364;
©	copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;

# Anchor Tag

The **< a >** tag defines a hyperlink, which is used to link from one page to another.

## Types of Hyperlinking:

- **External Hyperlinking:** links that point to a separate document (To create a link to another document, by using href attribute)
- **Internal Hyperlinking:** links that point to content within the same document.(To create a bookmark inside a document, by using the name attribute.)

Attribute	Description
href	Specifies the URL of the page the link goes to
target	_blank, _parent, _self, _top, framename
download	Specifies that the target will be downloaded when a user clicks on the hyperlink
type	Specifies the media type of the linked document
accessKey	Specifies a shortcut key to activate/focus an element [Alt] + accesskey

# Marquee Tag

An HTML <marquee> is a scrolling piece of text.

Attribute	Value	Description
behavior	scroll, slide, alternate	It defines the scrolling type.
direction	Up, down, left, right	It sets the direction for the scrolling content.
Height	pixels or %	pixels or % It defines the marquee's height.
loop	number	Specifies how many times to loop. The default value is INFINITE, which means that the marquee loops endlessly.
scrolldelay	seconds	Defines how long to delay between each jump.(seconds)
scrollamount	number	It defines the scrolling amount at each interval in pixels. Default value is 6.
bgcolor	rgb(x,x,x),#xxxxxx colorname	It is used to give a background color.

```
<marquee>
This is the example of marquee...
</marquee>
```

# Fieldset Tag

The **<fieldset>** tag is used to group related elements in a form.  
The **<fieldset>** tag draws a box around the related elements.

**<legend>** tag defines a caption for the **<fieldset>** element.

```
<fieldset>
  <legend> Personal Info: </legend>
  Name: <input type="text"><br/>
  Email: <input type="text">
</fieldset>
```

Personal Info:

Name:	<input type="text"/>
Email:	<input type="text"/>

# To Sum Up!!

- **What is HTML – HyperText Markup Language**
- **Why HTML is used – Used to create Web Page, easy, platform independent, support scripting language etc.**
- **HTML Page Structure - Three Sections (Comment, Head an Body)**
- **HTML Element & Attribute**
- **HTML Tag & Types of Tag - which provide instructions for how information will be processed or displayed on a web page.**
- **Basic HTML Tags - html, head, body, title, meta, Heading, paragraph, div, span, br, hr, pre etc.**
- **Text Formatting Tags – b, i, u, strong, sub, sup, big, small, font etc.**
- **List Tags – ol, ul and definition list**
- **Table – tr, td, th, caption and all attributes**
- **Img tag – To display image**
- **Anchor tag - for hyperlinking**
- **HTML Entities**
- **Marquee Tag**
- **FieldSet Tag**

# Quiz

## **Q1. It is used to display image**

- A. <image link="flower.jpg">
- B. 
- C. <img link="flower.jpg">
- D. <img source="flower.jpg">

**Answer: B**



## Q2. What is the correct HTML for creating a hyperlink?

- A. <a name="">A</a>
- B. <a>B</a>
- C. <a href="http://www.infowayltd.com">Infoway</a>
- D. <a url="http://www.infowayltd.com">Infoway</a>

Answer: C

<a href="http://www.infowayltd.com">Infoway</a>

**Q3. <b> tag makes the enclosed text bold. What is other tag to make text bold?**

- A. <strong>
- B. <dar>
- C. <black>
- D. <emp>

**Answer: A**

<strong>

#### **Q4. Dynamic web page**

- A. is same every time whenever it displays
- B. generates on demand by a program or a request from browser
- C. both (a) and (b)
- D. none of the mentioned

**Answer: B**

**generates on demand by a program or a request from browser**

## **Q5. Tag removed from the HTML 5.0**

- A. <P>
- B. <aside>
- C. <font>
- D. <nav>

**Answer: C**  
**<font> tag**

# Web Programming Technologies

## HTML Forms & HTML5 Elements Session-03

Harshita Maheshwari

# Agenda for Today's Session

- Form tag
- Form tag attributes
- DOM
- HTML Form Controls
- Validations for input elements
- iframe
- HTML5 New Elements
- Audio and Video
- Canvas
- HTML5 Geo-Location



# Form Tag

**<form> tag is used to create an HTML form for user input.**

The <form> element can contain one or more of the following form elements:

<b>&lt;input&gt;</b>	<b>&lt;textarea&gt;</b>	<b>&lt;button&gt;</b>
<b>&lt;select&gt;</b>	<b>&lt;fieldset&gt;</b>	<b>&lt;label&gt;</b>

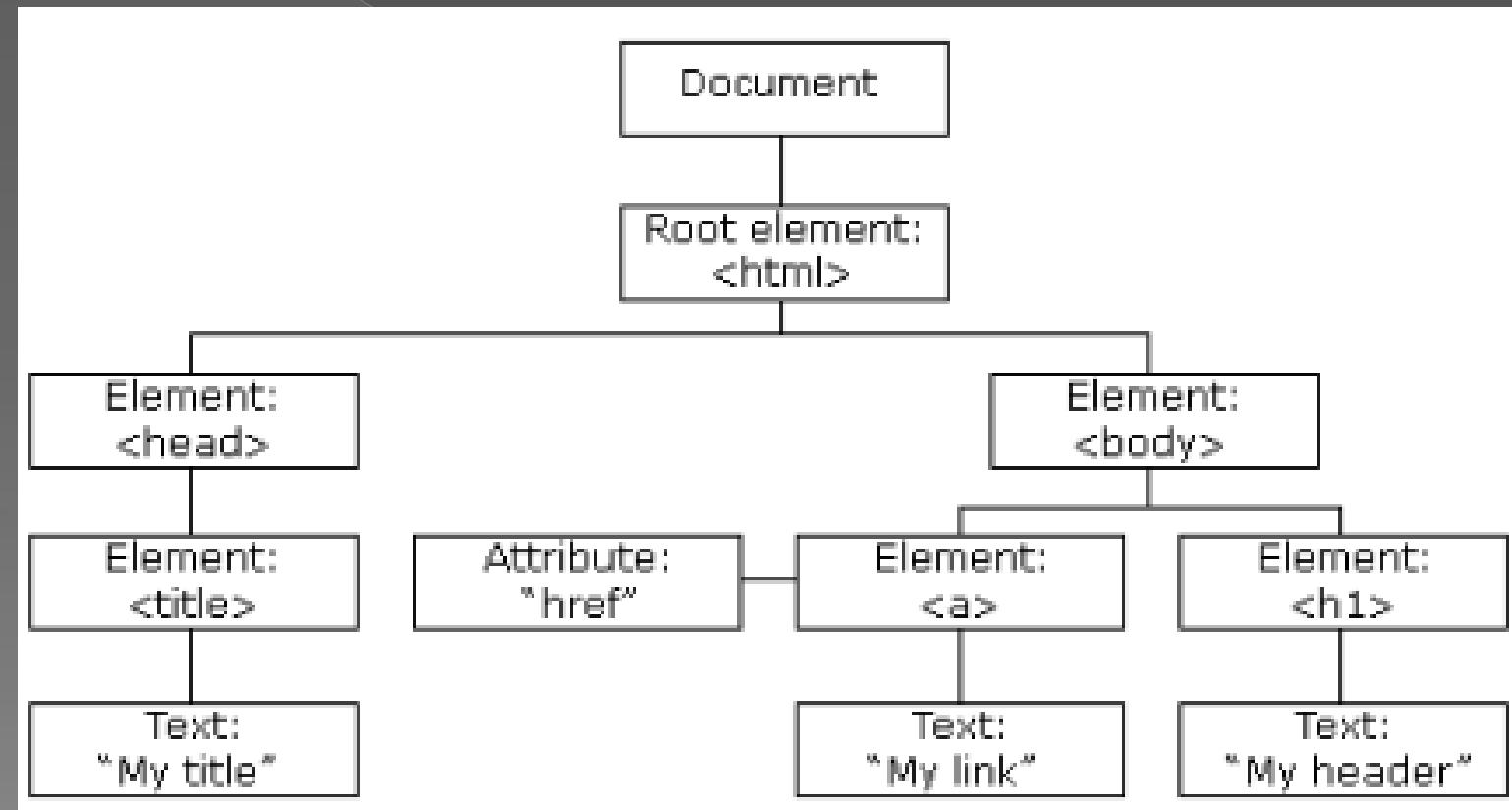
Attribute	Description
Action	URL
Method	Get, Post
Name	Specifies the name of a form
enctype	application/x-www-form-urlencoded, multipart/form-data text/plain
Autocomplete	On, off
novalidate	specifies that the form data should not be validated when submitted.
Target	_blank, _self, _parent, _top

GET	POST
Parameters remain in browser history because they are part of the URL	Parameters are not saved in browser history.
Can be bookmarked.	Can not be bookmarked.
application/x-www-form-urlencoded  only ASCII characters allowed.	multipart/form-data or application/x-www-form-urlencoded Use multipart encoding for binary data.  No restrictions. Binary data is also allowed.
GET is less secure compared to POST because data sent is part of the URL. So it's saved in browser history and server logs in plaintext.	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs.
Restrictions on form data length	No restrictions
GET method should not be used when sending passwords or other sensitive information.	POST method used when sending passwords or other sensitive information.
GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.	POST method variables are not displayed in the URL.
Can be cached	Not cached

# HTML DOM

When a web page is loaded, the browser creates a **Document Object Model** of the page.

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



# Input tag

**<input>** tag specifies an input field where the user can enter data.

Attribute	Description
autofocus	Specifies that an <input> element should automatically get focus when the page loads
checked	Specifies that an <input> element should be pre-selected when the page loads
Name	Specifies the name of an <input> element
placeholder	Specifies a short hint that describes the expected value of an <input> element
readonly	Specifies that an input field is read-only
Required	Specifies that an input field must be filled out before submitting the form
Value	Specifies the value of an <input> element
Pattern	specifies a regular expression that the <input> element's value is checked against.
min and max	specify the minimum and maximum values for an <input> element
Type	specifies the type <input> element to display
id	Specifies the id of an <input> element

# Type attribute

Type attribute specifies the type <input> element to display.

The following are the type attribute values:

<b>Button</b>	<b>Checkbox</b>	<b>Color</b>	<b>Date</b>
<b>Datetime</b>	<b>Email</b>	<b>File</b>	<b>Hidden</b>
<b>Image</b>	<b>Month</b>	<b>Number</b>	<b>Password</b>
<b>Radio</b>	<b>Range</b>	<b>Reset</b>	<b>Search</b>
<b>Submit</b>	<b>Text</b>	<b>Time</b>	<b>url</b>

```
<form method="post" action="hello.html">
    First name: <input type="text" />
    Last name: <input type="text" />
    Password: <input type="password" />
    File Upload:<input type="file" />
    <input type="submit" value="Submit" />
</form>
```

# Label tag

**<label>** tag defines a label for an **<input>** element.

Attribute	Value	Description
For	Element_id	Specifies which form element a label is bound to

```
<label for="male">Male</label>
<input type="radio" name="gender" id="male" value="male">
```

# checkbox and radio type

```
<input type="checkbox" name="hobby1" value="Music"> Music<br>
```

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

# File type

<b>Attribute</b>	<b>value</b>	<b>Description</b>
Accept	audio/* video/* image/* media_type	returns the value of the accept attribute of the file upload button.
Multiple	multiple	Sets or returns whether a user is allowed to select more than one file in the file upload field

```
<input type="file" name="pic" accept="image/*">
```

```
<input type="file" name="img" multiple>
```

# Select Tag

<select> element is used to create a drop-down list.

<option> Returns a collection of all the options in a drop-down list

Property	Description
Multiple	Sets or returns whether more than one option can be selected from the drop-down list
Value	Sets or returns the value of the selected option in a drop-down list
required	Specifies that the user is required to select a value before submitting the form
Size	Defines the number of visible options in a drop-down list
name	Defines a name for the drop-down list

# Optgroup tag

The <optgroup> is used to group related options in a drop-down list.

```
<select>
  <option selected disabled>Select Option</option>
  <optgroup label="Group1">
    <option value="Option1">Option1</option>
    <option value="Option2">Option2</option>
  </optgroup>
  <optgroup label="Group2">
    <option value="Option3">Option3</option>
    <option value="Option4">Option4</option>
  </optgroup>
</select>
```

# Textarea Tag

Property	Description
Cols	Sets or returns the value of the cols attribute of a text area
Rows	Sets or returns the value of the rows attribute of a text area
maxLength	Sets or returns the value of the maxlength attribute of a text area
placeholder	Specifies a short hint that describes the expected value of a text area
Readonly	Specifies that a text area should be read-only
Required	Specifies that a text area is required/must be filled out

```
<textarea rows="4" cols="50">  
  
</textarea>
```

# Number and range type

Define a field for entering a number.

```
<input type="number" name="quantity" min="1" max="5">
```

Define a control for entering a number whose exact value is not important (like a slider control).

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

## Attributes:

**max - specifies the maximum value allowed**

**min - specifies the minimum value allowed**

**step - specifies the legal number intervals**

**value - Specifies the default value**

# Iframe Tag

**Tag specifies an inline frame.**

**An inline frame is used to embed another document within the current HTML document.**

## **Attributes:**

src

height

width

name

```
<iframe src="home.html"></iframe>
```

# Audio Tag

**<audio> tag defines sound, such as music or other audio streams.**

Attribute	Description
autoplay	Specifies that the audio will start playing as soon as it is ready
Controls	Specifies that audio controls should be displayed (such as a play/pause button etc)
Loop	Specifies that the audio will start over again, every time it is finished
Muted	Specifies that the audio output should be muted
Src	Specifies the URL of the audio file

**<audio controls>**

**<source src="media/CollegeDays.mp3">**

**</audio>**

# Video Tag

**<video>** tag specifies video, such as a movie clip or other video streams.

```
<video width="320" height="240" controls>
  <source src="media/collegedays.mp4" type="video/mp4">
</video>
```

# Address Tag

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

```
<address>
```

Written by:

```
    <a href="http://infowayltd.com/">Sachin Chougule</a>.<br />
```

Visit us at: [www.infowayltd.com](http://www.infowayltd.com) <br />

Commerce Center, Rambaug Colony <br />

Paud Road, Opp. Krishna Hospital, PUNE-411038 <br />

Maharashtra

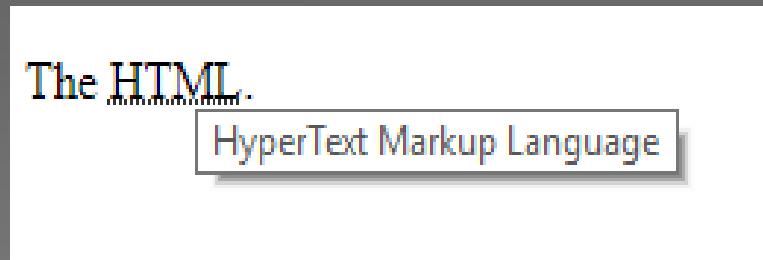
```
</address>
```

# Abbr Tag

**<abbr> tag defines an abbreviation or an acronym, like "Mr.", "Dec.", "ASAP", "ATM".**

**Ex.**

The <**abbr title="HyperText Markup Language">HTML</abbr>**.

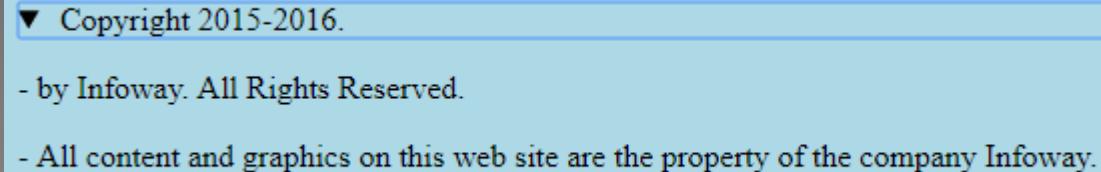


# Details Tag

<details> tag specifies additional details that the user can view or hide on demand.

<summary> tag defines a visible heading for the <details> element.

```
<details>
  <summary> Copyright 2015-2016.</summary>
  <p>- by Infoway. All Rights Reserved.</p>
  <p>- All content and graphics on this web site are the
    property of the company Infoway. </p>
</details>
```



▼ Copyright 2015-2016.  
- by Infoway. All Rights Reserved.  
- All content and graphics on this web site are the property of the company Infoway.

# nav tag

**<nav> tag defines a set of navigation links.**

```
<nav>
  <a href="#">Home</a>
  <a href="#">About Us</a>
  <a href="#">Contact Us</a>
</nav>
```

# canvas Tag

The HTML5 <canvas> tag is used to draw graphics, on the fly, with JavaScript.

The <canvas> element is only a container for graphics. we must use a script to actually draw the graphics.

Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

getContext() method returns an object that provides methods and properties for drawing on the canvas.

```
<canvas id="myCanvas" width="200" height="100" style="border:1px  
solid black;">  
</canvas>
```

# Geolocation tags

The HTML Geolocation API is used to get the geographical position of a user.

```
<script>
var x = document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}

function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
</script>
```

# To Sum Up!!

- **Form tag** - Used to create forms like signup, login
- **Form tag attributes** – methods(GET & POST), action, enctype etc.
- **DOM** – Tree Structure is created When a web page is loaded
- **HTML Form Controls** – input, select, textarea, label etc.
- **Validations for input elements** – readonly, checked, required etc.
- **Iframe** – inline frame
- **HTML5 New Elements** – abbr, address, details, summary etc.
- **Audio and Video**
- **Canvas** - for drawing
- **HTML5 Geo-Location** – current location (longitude and latitude)

# Quiz

**Q1. Which of the following tag automatically focus one particular form field in HTML5?**

- A. **output**
- B. **placeholder**
- C. **autofocus**
- D. **required**

**Answer: C**  
**autofocus**

**Q2 Which of the following method retrieves the current geographic location of the user?**

- A. geolocation.getCurrentPosition
- B. geolocation.watchPosition
- C. geolocation.clearPosition
- D. None of the above.

**Answer: A**

**geolocation.getCurrentPosition**

**Q3 \_\_\_\_\_ contains the navigation menu, or other navigation functionality for the page**

- A. **section**
- B. **header**
- C. **nav**
- D. **aside**

**Answer: C**  
**nav**

**Q4. Which of the following attribute is used to display date/time content?**

- A. **date**
- B. **datetime**
- C. **time**
- D. **year**

**Answer: B**  
**datetime**

**Q5. In html audio/video DOM, \_\_\_\_\_ sets or return whether the audio/video should be loaded when the page loads.**

- A. **autoplay**
- B. **buffered**
- C. **preload**
- D. **control**

**Answer: C**  
**preload**

# Thank you !!

# **Web Programming Technologies**

Harshita Maheshwari

# Session-3

# Topics to be covered.....

- Introduction to CSS
- Inserting CSS in an HTML Document
  - Internal Style Sheet
  - External Style Sheet
  - Inline Style Sheet
- CSS Selectors
- Pseudo Class & elements
- The CSS Box Model
- Font Properties
- Text Properties
- Position
- Background and border properties
- Display properties
- CSS Float

Cascading Style Sheets (CSS) form the presentation layer of the user interface.



CSS was introduced to keep the **presentation** information **separate** from **HTML** markup (content).

Tells the browser agent *how the* element is to be presented to the user.

# Before CSS

Initially Designers used presentation tags like (FONT, B, BR, TABLE etc.) to control the design of web pages.

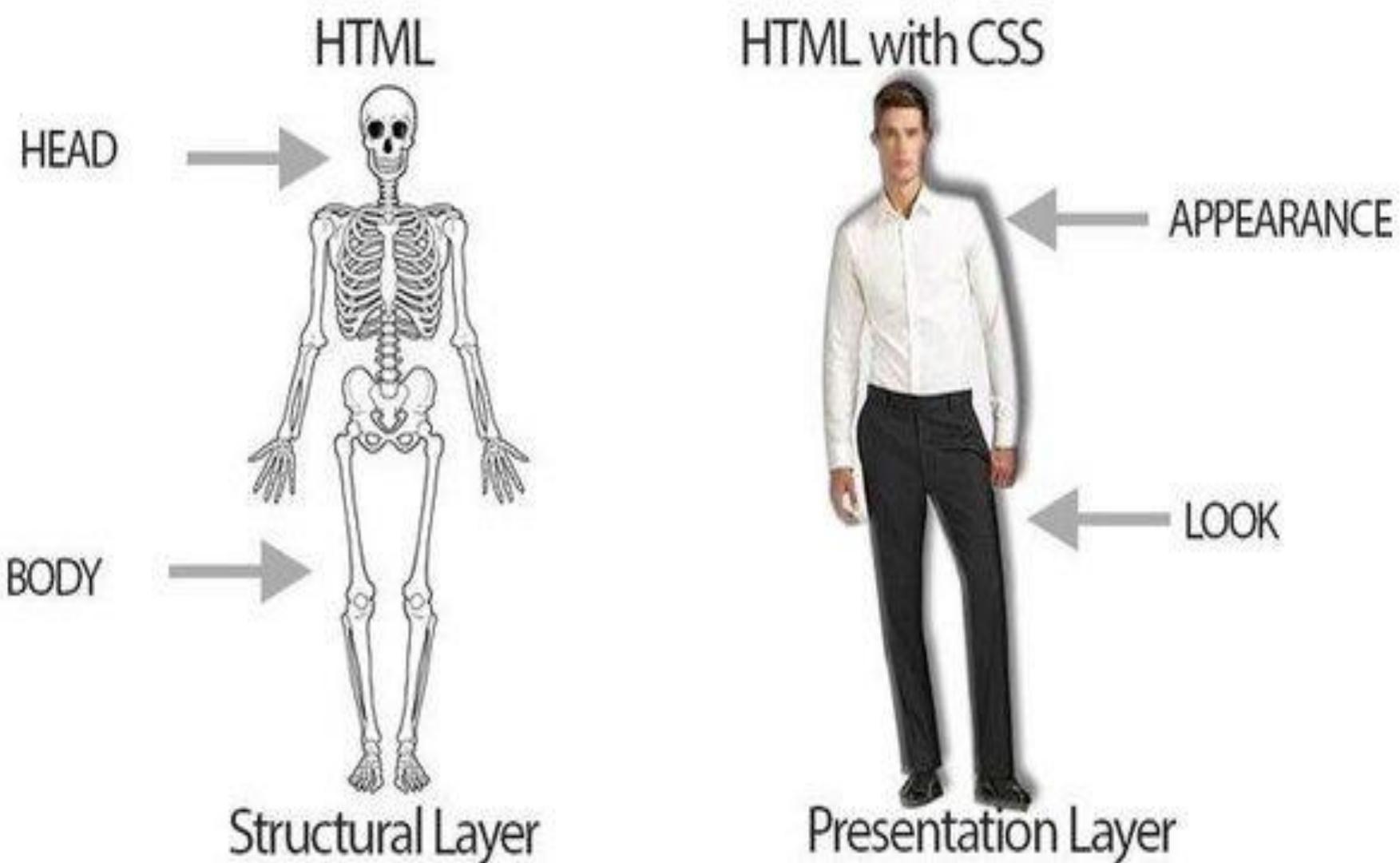
Any **modification** in the design of websites was a very **difficult** and **boring** task , as it evolves **manually editing** every HTML page.



```
<font size="14px">  
My First Header  
</font>  
<font size="12px" color="red"  
face="Verdana">  
My information 1 goes here.  
</font>  
<font size="14px">  
My Second Header  
</font>  
<font size="12px" color="red"  
face="Verdana">  
Different information goes here.  
</font>
```

*“HTML without CSS is like a piece of candy without a pretty wrapper.”*

- What can we do with CSS that we can't do with HTML?
  - Control of backgrounds.
  - Set font size to the exact height you want.
  - Highlight words, entire paragraphs, headings or even individual letters with background colors.
  - Overlap words and make logo-type headers without making images.
  - Precise positioning.
  - Linked style sheets to control the look of a whole website from one single location.
  - And more.



A CSS RULE is made up of a selector and a declaration. A declaration consists of property and value.

selector {property: value; }  
declaration

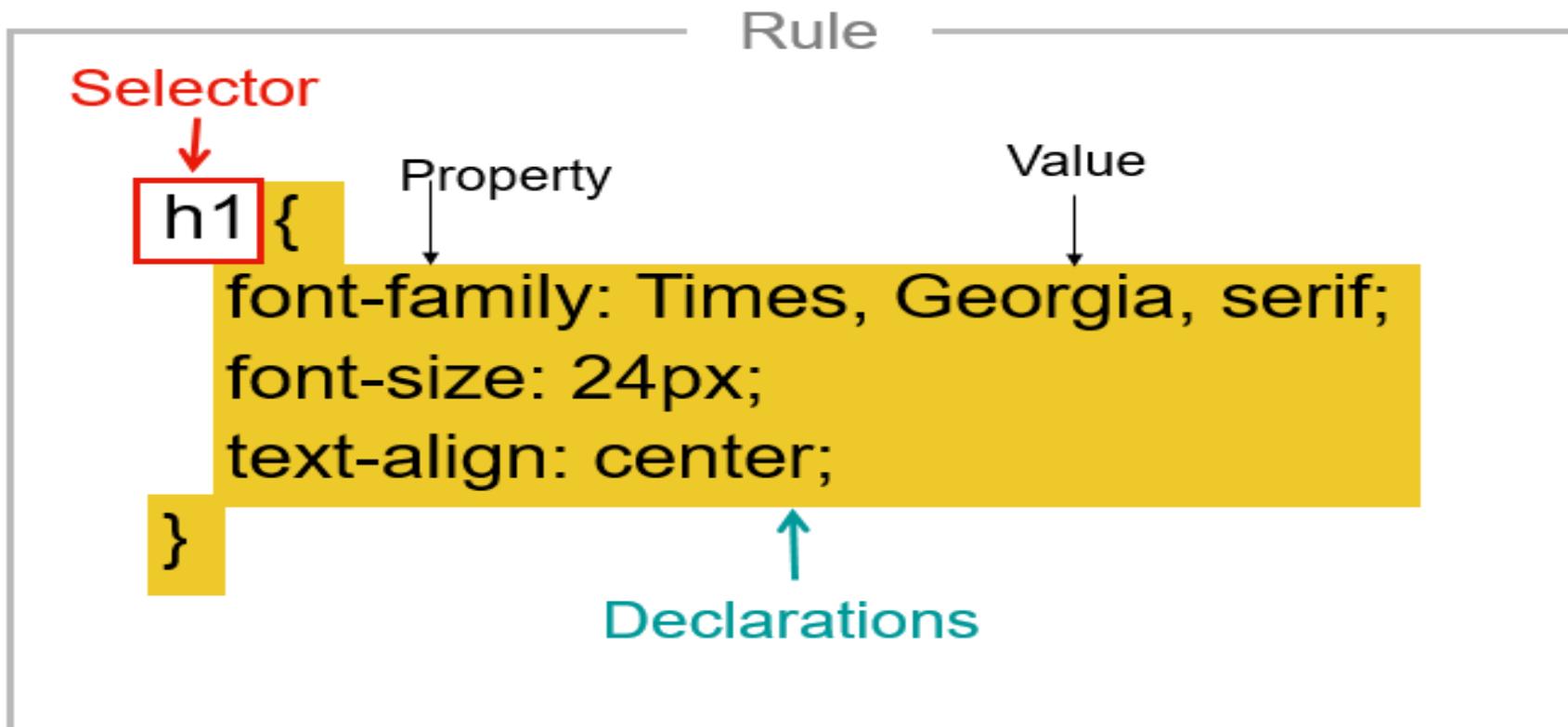
what {  
is : CSS ;  
}

**Selector:** A selector is an HTML tag at which style will be applied.

**Declaration:** enclosed within { }.

Declaration has two sections separated by colon(:).

Properties and values tell an HTML element how to display.



- 1)Local (Inline Stylesheet )
- 2)Global (Embedded/Internal Stylesheet )
- 3)Linked (External Stylesheet )
  - a)Linking to External Stylesheet
  - b) Importing to External Stylesheet

- **Inline** style sheet.
- Placed inside tags.
- Specific to a single instance of an html tag on a page.
- It is implemented by using style attributes with the HTML tag.

Example:

```
<p style="font-size: 10pt; color: red; font-weight: bold;  
font-family: Arial, Helvetica, sans-serif">
```

This is a local stylesheet declaration. </p>

On the browser: →

This is a local stylesheet declaration.

- **Embedded** or **internal** style sheet
- Applicable to an entire document
- **Internal styling** is defined in the **<head>** section of an HTML page, within a **<style>** element.

```
<html>
<head>
    <style type="text/css">
        h1 {
            background-color:green;
            color:yellow;
        }
    </style>
</head>
<body>
    <h1> Infoway Technologies, PUNE </h1>
</body>
</html>
```

- **External** style sheet
- Styles are saved in a separate file, with the extension **.css**
- This single stylesheet can be used to define the look of multiple pages.

```
p {font-family: verdana, sans-serif;  
font-size: 12pt; color: red}
```

```
h1 {font-family: serif; font-size:  
14pt; color: green}
```

```
h2 {font-family: serif; font-size:  
11pt; color: blue}
```

Save this text  
file as **style.css**

To apply the stylesheet “***style.css***“ to an HTML document, call it in from the header:

```
<head>
  <link rel="stylesheet" href="style.css"
        type="text/css">
</head>
```

# Importing to an External Stylesheet

The @import rule is another way of loading an external style sheet. The @import statement instructs the browser to load an external style sheet and use its styles.

```
<head>
<style>
    @import "style.css";
</style>
</head>
<Body>
    <p>this example shows the Importing to an external stylesheet</p>
</body>
```

We can use the @import rule to import a style sheet within another style sheet.

```
@import url("css/layout.css");
@import url("css/color.css");
body {
    color: blue;
    font-size: 14px;
```

# Selector

Universal selectors are used to select any element.

```
* {  
color: blue;  
}
```

- Tag (Tag name)

**HTML**

```
<div>
    Text
</div>
<div>
    <span>some text </span>
</div>
<span>some other text </span>
```

**CSS**

```
DIV {
    width: 200px;
}
SPAN {
    font-size:130%;
}
```

ID selectors should be used with **single** elements.

### HTML

```
<div id="content">  
    Text  
</div>
```

### CSS

```
#content {  
    width: 200px;  
}
```

# Class based selector (.)

Class based styles can be used by **multiple** **HTML elements.**

## HTML

```
<div class="big">  
    Text  
</div>  
<div>  
    <span class="big">some text </span>  
</div>
```

## CSS

```
.content {  
    width: 200px;  
}
```

The most important difference between IDs and classes is that there can be only one ID on a page, but multiple classes.

An ID is more specific than a class.

An element can have both an ID and multiple classes.



**ID: #344-34-4344**

**Class: Male**

**Class: Employee**



**ID: #123-54-9877**

**Class: Female**

**Class: Employee**

# Grouping Selectors

**Group different selectors with the same declaration on one line.**

```
h1, h2, h3 {color: yellow;}
```

# Descendant selectors

Descendant selectors are used to select elements that are descendants (**not necessarily children**) of another element in the document tree.

```
<ul>
  <li>PGDAC</li>
  <li>Infway Courses
    <ol>
      <li>PreDAC</li>
      <li>Certification courses</li>
    </ol>
  </li>

</ul>
```

```
<style>
  ul li{color:blue}
</style>
```

- PGDAC
- Infway Courses
  - 1. PreDAC
  - 2. Certification courses

# Child selectors

A child selector is used to select an element that is a direct child of another element (parent). Child selectors will not select all descendants, only direct children.

```
<div>
    <h2>Infoway Technologies</h2>
    <p>Kothrud,Pune.</p>
    <center> <h2>Authorized C-DAC Training Centre in Pune </h2></center>
</div>
```

---

```
<style>
    div>h2{background-color: yellowgreen}
</style>
```

**Infoway Technologies**

Kothrud,Pune.

**Authorized C-DAC Training Centre in Pune**

# Adjacent sibling selectors

Adjacent sibling selectors will select the sibling immediately following an element.

Syntax:

```
element + element {  
    css declarations;  
}
```

# General sibling selectors

The general sibling selector selects all elements that are siblings of a specified element.

Syntax:

```
element ~ element {  
    css declarations;  
}
```

Attribute selectors selects elements based upon the attributes present in the HTML Tags and their value.

```
IMG [src="small.gif"] {      will work for  
border: 1px solid #000;        
}
```

CSS pseudo-classes are used to add special effects to some selectors.

Syntax:      selector:pseudo-class {  
                                  property: value;  
                                  } }

Ex. the state of being hovered, or the state of being activated.

```
a:hover {  
               color: red;  
}
```

:link	:active	:visited	:hover	:target
:focus	:enabled	:disabled	:required	:readonly
	:valid	:invalid	:checked	
:first-child	:last-child	:only-child		

A CSS **pseudo-element** is a keyword added to a selector that lets you style a specific part of the selected element(s).

Syntax:      `selector::pseudo-element {  
                  property: value;  
                  }  
}`

Ex. `::first-line` can be used to change the font of the first line of a paragraph.

```
p::first-line {  
                  color: blue;  
                  text-transform: uppercase;  
                  }
```

<code>::after</code>	<code>::before</code>
<code>::first-letter</code>	<code>::first-line</code>
<code>::selection</code>	

# Properties

- **Font-family:** specifies the font for an element.

- **Font-size:** The size of the font

- **Font-weight**

This states whether the text is bold or not.

Commonly used are font-weight: bold or fontweight: normal. In theory it can also be bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800 or 900

- **Font-style**

font-style: *italic* or

font-style: normal.

- **Text-decoration**

This states whether the text is underlined or not.

- *text-decoration: overline*

- *text-decoration: line-through, strike-through,*

- *text-decoration: underline ( should only be used for links )*

This property is usually used to decorate links, such as specifying no underline with

- *text-decoration: none.*

- ***text-transform***

This will change the case of the text.

text-transform: capitalize

text-transform: uppercase

text-transform: lowercase

text-transform: none

- ***Text spacing***

letter-spacing and word-spacing

line-height

text-align

text-indent

Eg.

```
p {  
letter-spacing: 2px;  
word-spacing: 3px;  
line-height: 4px;  
text-align: center;  
}
```

# Properties - Background

- Background-Attachment
  - fixed,scroll
- background-color
  - color name,hexadecimal number,RGB color code,transparent
- background-image:
  - url(path\_to\_image), linear-gradient(),radial-gradient(),  
repeating-linear-gradient(),repeating-radial-gradient()
- background-position
  - top left,top center,top right,center left,center center,center  
right,bottom left,bottom center,bottom right
- Background Repeat
  - no-repeat,Repeat,repeat-x,repeat-y

# Properties - Border

- border-color
  - color
- border-style

Text with solid border.

Text with double border.

Text with groove border.

Text with dotted border.

Text with dashed border.

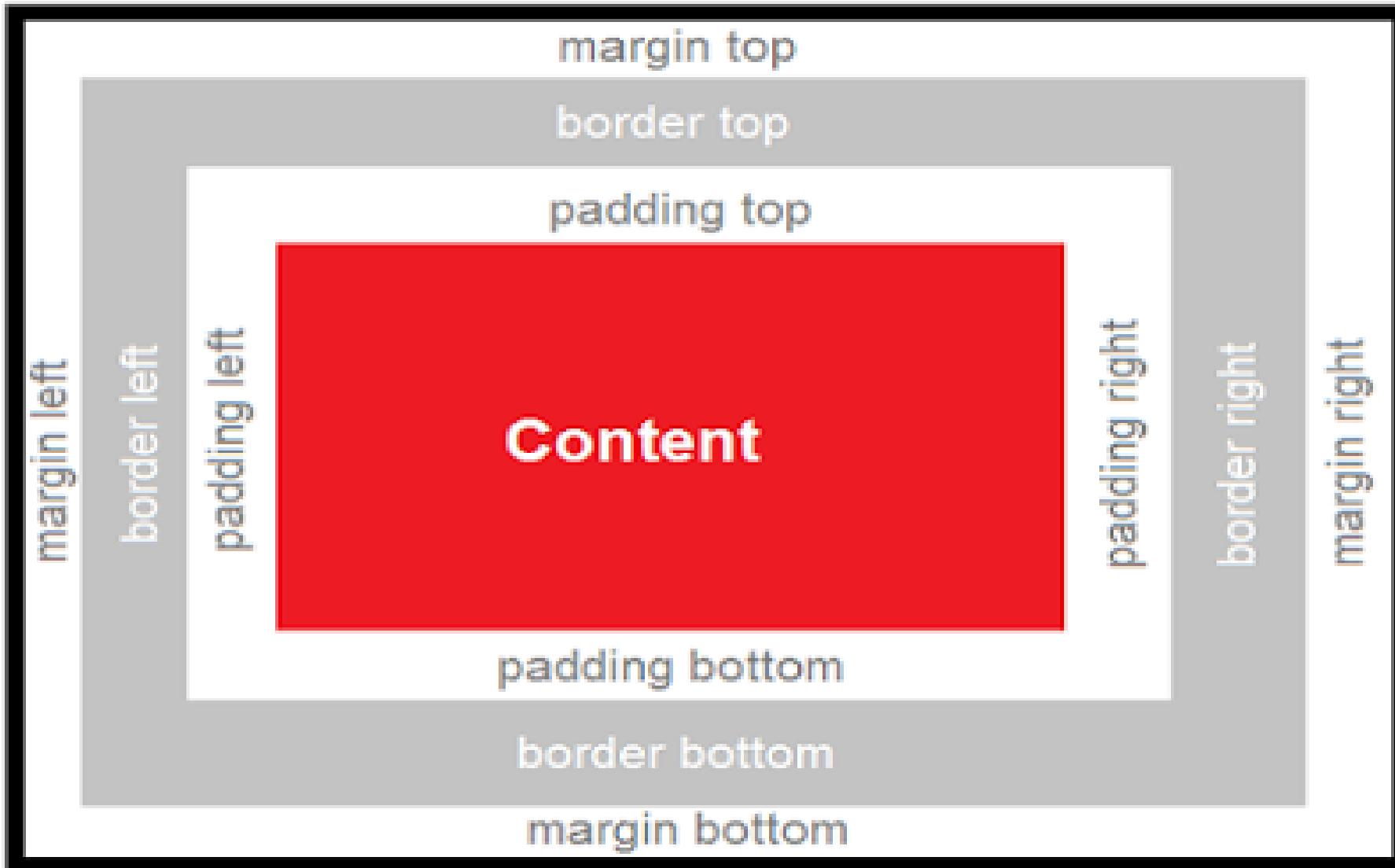
Text with inset border.

Text with outset border.

Text with ridge border.

Text with hidden border.

- border-width
  - Length,Thin,Medium,Thick
- border-bottom: 1px solid red;
- border-left: 1px solid red;
- border-right: 1px solid red;
- border-top: 1px solid red;
- border-radius
  - length,percentage
- border-collapse
  - separate,collapse



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

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

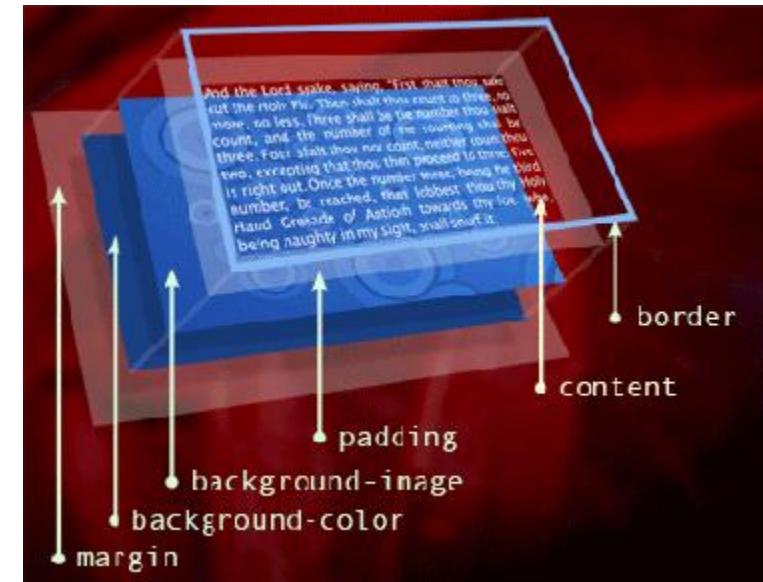
# Margins and Padding

Margins and Padding are the two most commonly used properties for **spacing-out elements**.

A margin is the space **outside of the element**, whereas **padding is the space inside the element**.

Eg:

```
h2 {  
font-size: 10px;  
background-color: #1F488D;  
margin: 10px;  
padding: 5px;  
}
```



# Padding

- `#sample { padding: 10px; }`
- `#sample { padding: 10px 5px }`
- `#sample { padding: 10px 5px 2px }`
- `#sample { padding: 10px 5px 2px 5px }`
- one value, such as 10px, to specify equal padding on every side
- two values, such as 10px 5px, to specify top/bottom (first value) and right/left (second value) padding
- three values, such as 10px 5px 2px, to specify top (first value), right/left (second value) and bottom (third value) padding
- four values, such as 10px 5px 2px 1px to specify top, right, bottom and left padding respectively
  - `padding-left: 10px;`
  - `padding-right: 10px;`
  - `padding-bottom: 10px;`
  - `padding-top: 10px;`

- **None : The element will not be displayed.**

```
p {display : none}
```

- **Block : The element will be displayed as a blocklevel element, with a line break before and after the element.**

```
p {display : block}
```

- **Inline : The element will be displayed as an inline element, with no line break before or after the element.**

```
p {display : inline}
```

**Visible** : The element is visible (default).

**Hidden** : The element is invisible (but still takes up space)

This is small text and **this is big** I am Italic

```
.big {  
    visibility:hidden;  
}
```

This is small text and \_\_\_\_\_ I am Italic

Float property makes elements float to the right or left of the screen, positioned where they are in the HTML.

With the increase of laptop computer, traditional Desktop computer are slowly getting removed from the market.



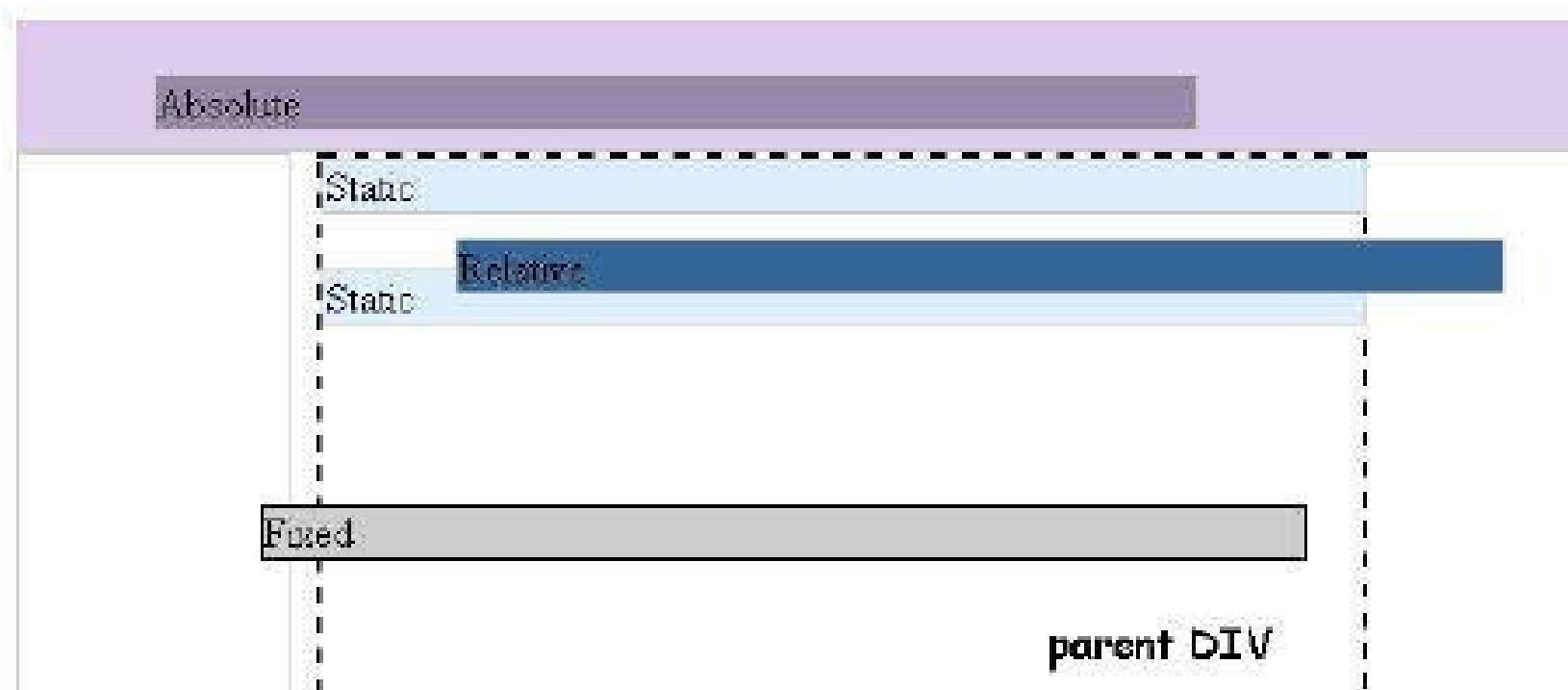
The ease of taking laptop anywhere with you is the biggest advantage of the Laptop computers. Laptop computers are still double as costly than desktop computers.

```
IMG  
{   float:  
left;  
}
```

With the increase of laptop computer, traditional Desktop computer are slowly getting removed from the market. The ease of taking laptop anywhere with you is the biggest advantage of the Laptop computers. Laptop computers are still double as costly than desktop computers.

- list-style
  - image,Position,type
- list-style-image: url(path\_to\_image.gif, jpg or png);
- list-style-position: value;
  - inside
    - Coffee - A brewed drink prepared from roasted coffee beans...
    - Tea
    - Coca-cola
  - outside
    - Coffee - A brewed drink prepared from roasted coffee beans...
    - Tea
    - Coca-cola
- list-style-type: value;
  - disc,Circle,Square,Decimal,lower-roman,upper-roman,lower-alpha,upper-alpha,none

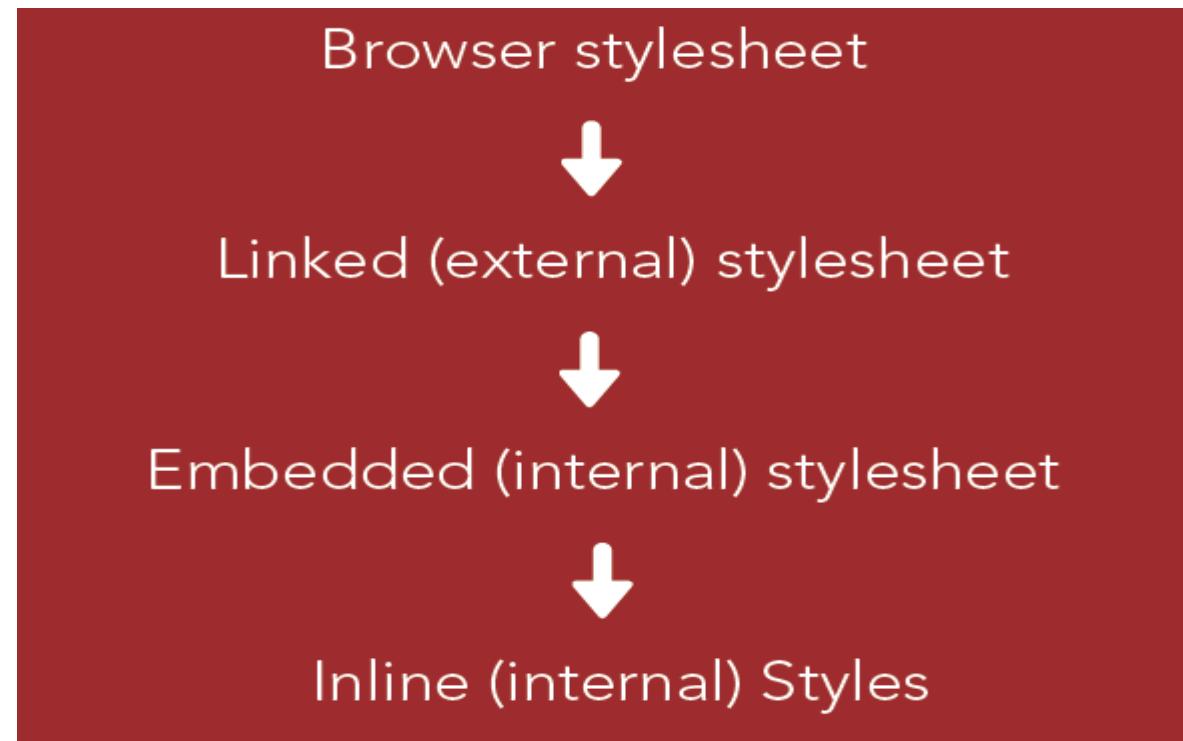
- **Static**
- **Relative**
- **Absolute**
- **Fixed**



- Static: The value static is the default value for elements and renders the position in the normal order of things as they appear in the html
- Relative: It is much like static, but the element can be offset from its original position with the properties top, right, bottom and *left* .
- Absolute: The absolute element can be placed anywhere on the page using top, right, bottom and *left* .
- Fixed: It behaves like absolute, but fixed elements should stay exactly where they are on the screen even when the page is scrolled.

The “cascade” part of CSS is a set of rules for resolving conflicts with multiple CSS rules applied to the same elements.

For example, if there are two rules defining the color of your h1 elements, the rule that comes last in the cascade order will “trump” the other.



- Inline (local) overrides internal (global)
- Internal (global) overrides external (linked).
- An inline style (inside an HTML element) has the highest priority, which means that it will override every style declared inside the `<head>` tag, in an external style sheet, and in the browser (default value).

Most elements will inherit many style properties from their parent elements by default.

HTML	relationship
<body>	parent of site
<div>	parent of ul and li, child of body
<ul>	parent of li, child of div and body
<li></li>	child of ul, div, and body
</ul>	
</div>	
</body>	

**body**

make the paragraph 16px, Verdana, red

---



**p**

make the paragraph blue

---



16px, Verdana, blue

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <style>
        div {
            background-color: green !important;
        }
    </style>
</head>
<body>
    <div style="background-color:red">
        <p>This is the example of important property</p>
    </div>
</body>
</html>
```

# Thank You

# **Web Programming Technologies**

Harshita Maheshwari

# JavaScript



# Session-1

# Topics to be covered...

- What is Scripting Language
- Client vs. Server Side Scripting
- Introduction to JavaScript
- Where JavaScript is used
- Variables in Javascript
- JavaScript Console
- Let keyword
- Use Strict Keyword
- Javascript Hoisting
- JS Operators
- Function
- Control Structure and loops
- Write(),alert(),confirm() and prompt() box
- DOM Events
- Global Properties and methods

A Script is a program or sequence of instructions that is interpreted or carried out by another program rather than by computer processor.

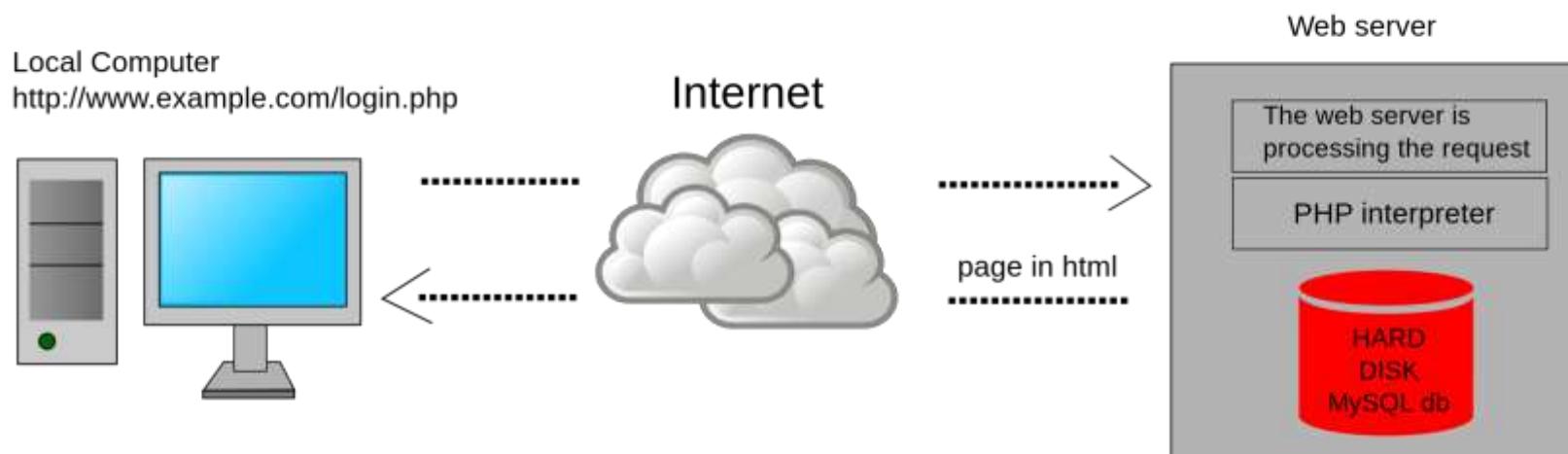
The scripting language is basically a language where instructions are written for a run time environment. They do not require the compilation step and are rather interpreted.

# Server Side Scripting

Server side scripting is used to connect to the database that reside on the web server.

Server side scripting can access the file system residing at the web server.

Response from a server-side script is slower as compared to a client-side script because the scripts are processed on the remote computer.



Advantage: Your scripts are hidden from view. Users only see the HTML output , even when they view the source.

# Client Side Scripting

Program that execute on client side, by the web browser instead of server side.

Upon request, the necessary files are sent to the user's computer by the web server on which they reside.



Advantages:

Allow for more interactive by immediately responding to user actions.

Execute quickly because they don't require a trip to the server.

Can give developers more control over the look and behavior of their web apps.

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

Tells where the JavaScript starts

Commands for writing output to a page

Tells where the JavaScript ends

This code produce the output on an HTML page:  
**Hello World!**

# What is ECMAScript?

ECMAScript (European Computer Manufacturers Association ) is a scripting language standard and specification

JavaScript

Jscript

ActionScript

# What is JavaScript?

JavaScript is used to program the behaviour of web pages (performing dynamic tasks).

JavaScript are scripts (code) that is executed on the client's browser instead of the web-server (Client-side scripts).

JavaScript is lightweight and cross-platform and loosely coupled.

**JavaScript was created by Brendan Eich, a Netscape Communications Corporation programmer, in September 1995.**



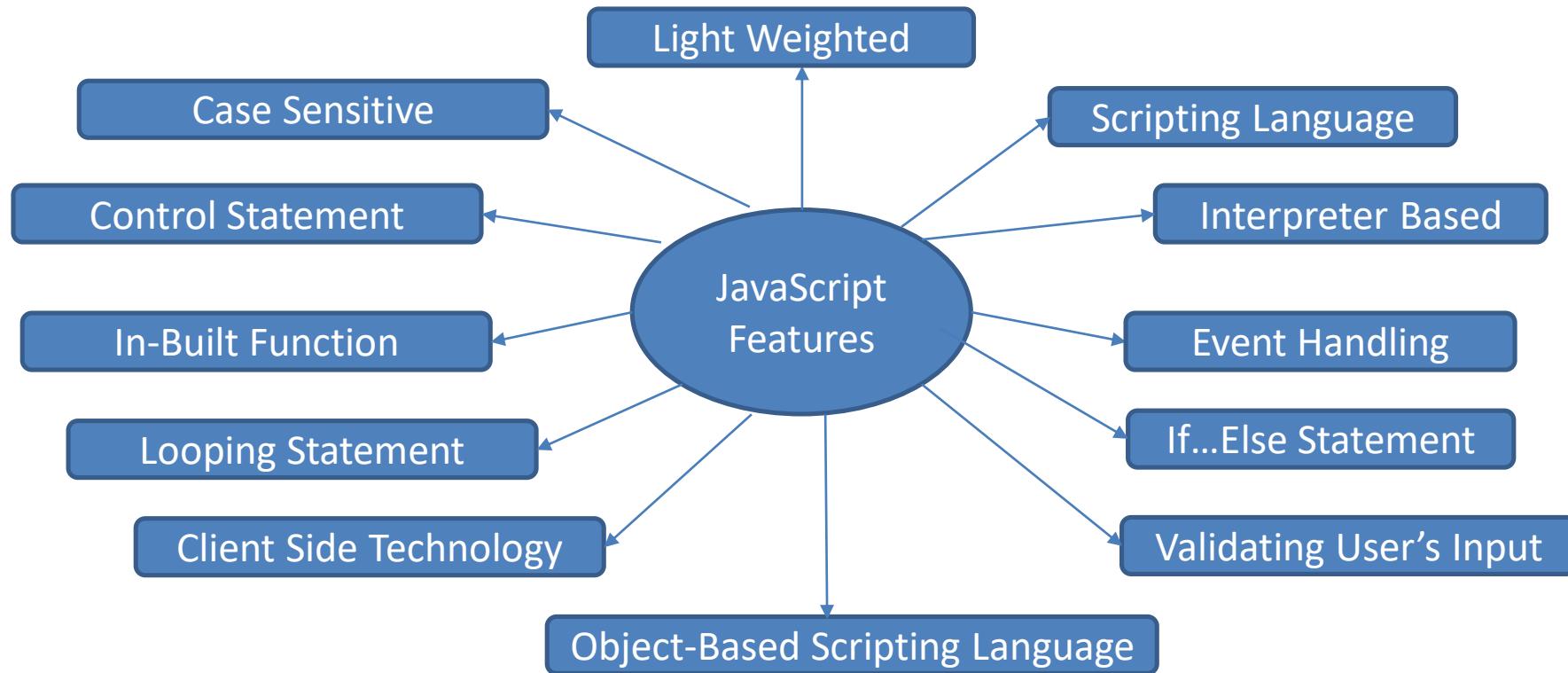
- **HTML**
  - Hypertext Markup Language
  - Structure of Page
- **JavaScript**
  - Interactivity with User
  - Dynamic Updates in a Web Page
- **CSS**
  - Cascading Style Sheets
  - Presentation/Styling



# Why we need client side programming

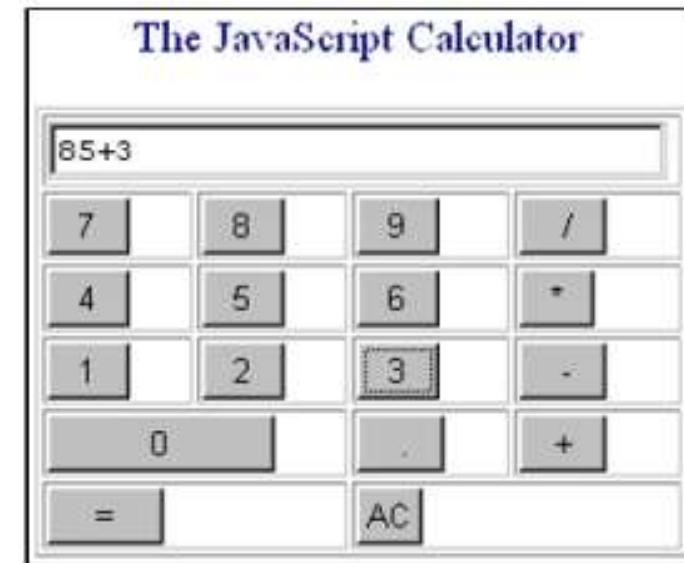
- The user's actions will result in an immediate response because they don't require a trip to the server.
- Allows the creation of faster and more responsive web applications.
- Make web pages more interactive
- Fewer resources are used and needed on the web-server.

# JavaScript Features



# What Javascript can do?

- Javascript can change HTML Content
- Javascript can change HTML Attributes
- Javascript can change HTML Styles (CSS)
- Javascript can validate Data
- Javascript can Make Calculations



# Embedding JavaScript in HTML

1. Anywhere in the html file between <script></script> tags.

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

<h2>JavaScript Demo</h2>

<p id="demo"></p>

<script>
document.write("My First JavaScript");
</script>
|
</body>
</html>
```

2. In an external file and refer to it using the SRC attribute.

```
<!DOCTYPE html>
<html>
<head>
<title>External Javascript</title>
<script src="myScript.js"></script>
</head>
<body>
<p id="demo">A Paragraph.</p>
</body>
</html>
```

# JavaScript Display Possibilities

JavaScript can "display" data in different ways:

1. Writing into the browser console, using **console.log()**.
2. Writing into the HTML output using **document.write()**.
3. Writing into an alert box, using **window.alert()**.
4. Writing into an HTML element, using **innerHTML**.

# Console Object

The Console object provides access to the browser's debugging console.

## Console Object Methods:

**log() method:** writes a message to the console.

Syntax

`console.log(message)`

```
<!DOCTYPE html>
<html>
<body>
<script>
console.log("Hello world!");
</script>
</body>
</html>
```

**table() Method:** writes a table in the console view.

```
<!DOCTYPE html>
<html>
<body>
<script>
console.table(["Audi", "Volvo", "Ford"]);
</script>
</body>
</html>
```

(index)	Value
0	"Audi"
1	"Volvo"
2	"Ford"

▶ Array(3)

```
<!DOCTYPE html>
<html>
<body>
<script>
var car1 = { name : "Audi", model : "A4" }
var car2 = { name : "Volvo", model : "XC90" }
var car3 = { name : "Ford", model : "Fusion" }

console.table([car1, car2, car3]);
</script>
</body>
</html>
```

(index)	name	model
0	"Audi"	"A4"
1	"Volvo"	"XC90"
2	"Ford"	"Fusion"

▶ Array(3)

**clear() Method:** clears the console.

The console.clear() method will also write a message in the console: "Console was cleared".

Syntax

console.clear()

# Alerts

An alert box is often used if you want to make sure information comes through to the user.

Syntax

```
window.alert("sometext");
```

```
alert("I am an alert box!");
```

An embedded page on this page says

I am an alert box!

OK

# Prompts and Confirm

Prompts :The return is the data the user entered

```
<script type="text/javascript">  
window.prompt('Message', 'Initial Value');  
</script>
```

Confirm: The confirm returns true and false

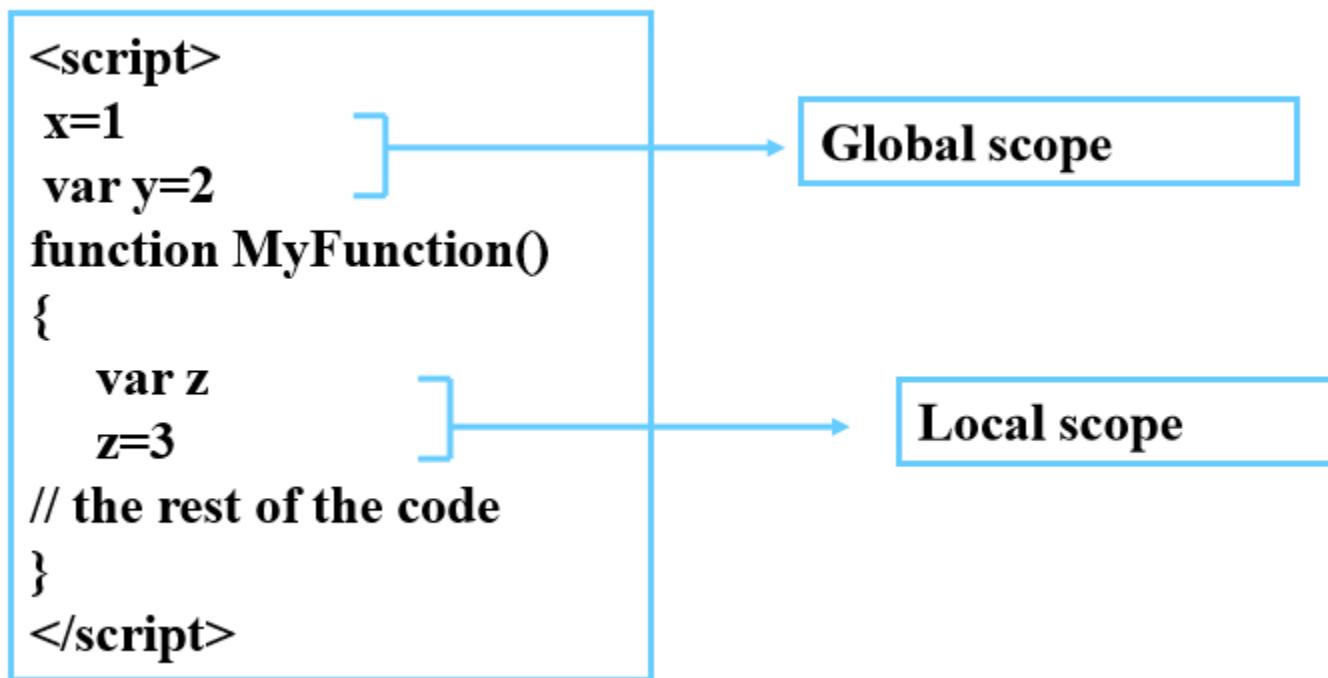
```
<script type="text/javascript">  
window.confirm('Message');  
</script>
```

# Variables

JavaScript variables are containers for storing data values..

Variables are declared with the '**var**' keyword.

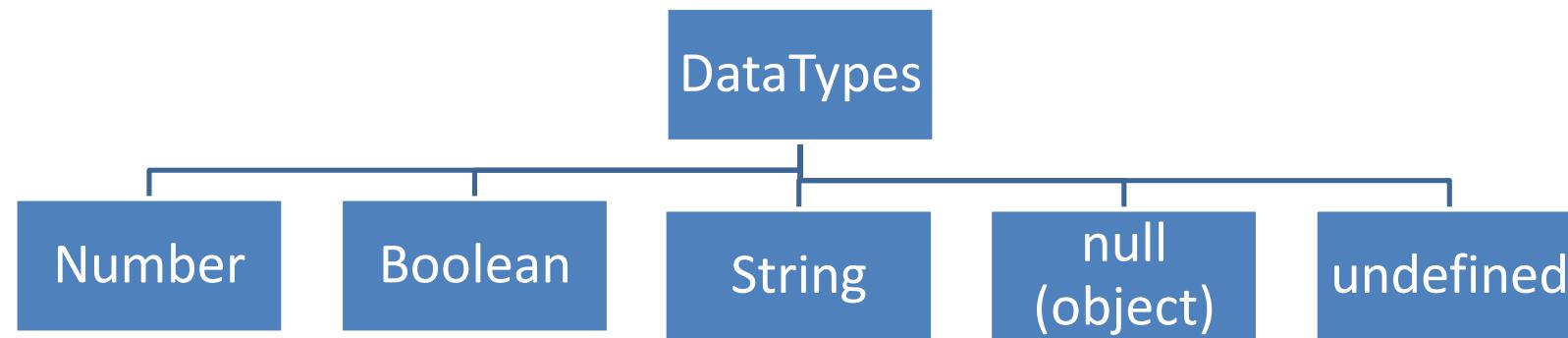
```
var a=10;
```



## Lexical Scoping:

Depending on position meaning changes.

# Data Types



```
var x=10;  
alert(typeof(x));
```

# undefined value and null value

**Undefined** value means the variable is declared but not assigned.

**Ex.**

```
var x;  
console.log(x);  
  
x=10;
```

**null** value means the variable is declared and assigned as null means noting neighter number nor string or boolean.

```
var x=null;  
console.log(x);
```

# Declaring are Assigning variables

```
var x=10; //Declare and Assigning
```

```
var t; //Declaring  
var t=20; //Assigning
```

```
y=10; //Declaring auto and Assigning
```

# "use strict"

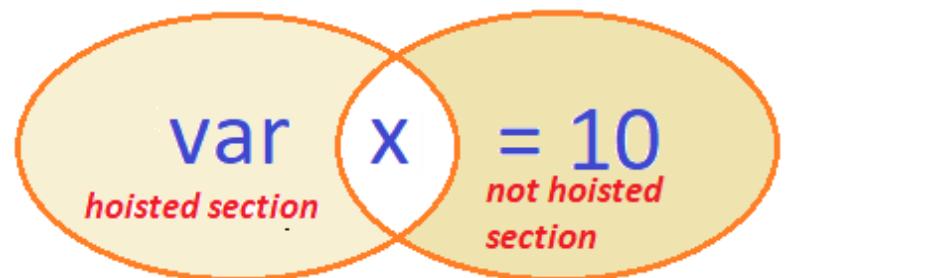
The strict mode in JavaScript does not allow following things:

- Use of undefined variables
- Use of reserved keywords as variable or function name
- Duplicate properties of an object
- Duplicate parameters of function
- Assign values to read-only properties
- Modifying arguments object

```
"use strict";
x = 3.14;      // This will cause an error because x is not declared
```

# JavaScript Hoisting

Hoisting is JavaScript's **default behavior of moving all declarations to the top of the current scope** (to the top of the **current script** or the **current function**).



Declaration  
moves  
To top

```
<script>
    function f1()
    {
        var x;
        alert(x);
        if(1==1)
        {
            var x=10;
        }
    }
    f1()
</script>
```

The diagram shows a transformation of JavaScript code. On the left, the original code is shown within a script tag. It contains a function declaration "function f1()", followed by a block of statements. Inside the block, there is a "var x;" declaration, an "alert(x);" call, a conditional statement "if(1==1)", another "var x=10;" declaration, and a closing brace "}". An arrow points from this code to the transformed version on the right. In the transformed version, the entire declaration "var x;" has been moved to the very top of the function block, before the first statement.

# Javascript Operators

**typeof:** operator returns a string indicating the type of the unevaluated operand.

Syntax:

`typeof operand`

The following table summarizes the possible return values of `typeof`

Type	Result
Undefined	"undefined"
Null	"object"
Boolean	"boolean"
Number	"number"
String	"string"

# Arithmetic Operator

## Addition (+) Operator:

The addition operator produces the sum of numeric operands or string concatenation.

Type	Result
Number + Number (addition)	$10 + 10 = 20$
Boolean + Number (addition)	<code>true + 10 = 11</code>
Boolean + Boolean (addition)	<code>true + true = 2 / false + false = 0</code>
Boolean + Boolean + Number (addition)	<code>true + true + 1 = 3 / true + false + 1 = 2</code>
String + String (concatenation)	<code>"Infoway" + "Kothurd" = InfowayKothrud</code>
String + Number (concatenation)	<code>'Infoway' + 10 = Infoway10</code>
String + Boolean (concatenation)	<code>"A" + true = Atrue</code>
Number + String (concatenation)	<code>10 + "Infoway" = 10Infoway</code>
Number + String + Number (concatenation)	<code>10 + 'Infoway' + 10 = 10Infoway10</code>
Number + Number + String (addition & concatenation)	<code>10+10 + "Infoway" = 20Infoway</code>
Boolean + Boolean + String (addition & concatenation)	<code>true + true + "Infoway" = 2Infoway</code>

## Subtraction (-) Operator

Type	Result
Number - String	10 - 'A' = NaN
String - Number	"A" - 10 = NaN

## Multiplication (\*) Operator

Type	Result
Number * String	10 * 'A' = NaN
String * Number	"A" * 10 = NaN

## Division (/) Operator

Type	Result
Number / String	10 / 'A' = NaN
Number / Number	10 / 0 = Infinity

# Comparisons operators

Operator	Description
<code>==</code>	equal to
<code>====</code>	equal value and equal type
<code>!=</code>	not equal
<code>!==</code>	not equal value or not equal type
<code>&gt;</code>	greater than
<code>&lt;</code>	less than
<code>&gt;=</code>	greater than or equal to
<code>&lt;=</code>	less than or equal to
<code>?</code>	ternary operator

# Equality (==) operators

The **equality** (==) operator converts the operands if they are not of the same type, then applies strict comparison.

Syntax:

A==B

```
1 == 1          // true
'1' == 1        // true
1 == '1'        // true
0 == false      // true
0 == null       // false
```

# Identity / strict equality (==) operators

The **identity** (==) operator returns true if the operands are strictly equal with no type conversion.

Syntax:

A==B

```
1 === 1      // true
```

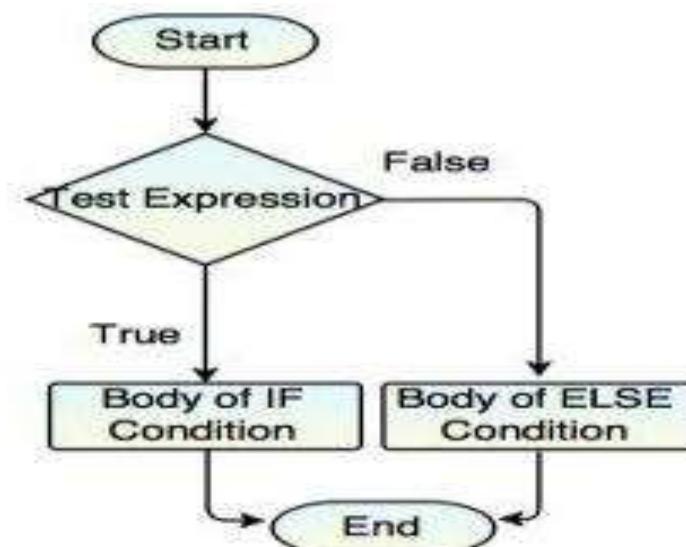
```
1 === '1'    // false
```

# if Statements

**if-else statement** is used to execute the code whether condition is true or false.

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

}
Else
{
    //Statement 2;
}
```



# Switch Statement

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

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

# Loops

## for Loop:

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

## While Loop

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

## Do-While Loop

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

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

# Events

Events are actions or occurrences that happen in the system you are programming, which the system tells you about so you can respond to them in some way if desired.

For example, if the user clicks a button on a webpage, you might want to respond to that action by displaying an information box.

```
<script type="text/javascript">
  function click_event()
  {
    document.write("Button is clicked");
  }
</script>
</head>
<body>
<button onclick="click_event()">Click Me</button>
</body>
```

Events	Description
onclick	occurs when element is clicked.
ondblclick	occurs when element is double-clicked.
onfocus	occurs when an element gets focus such as button, input, textarea etc.
onblur	occurs when form loses the focus from an element.
onsubmit	occurs when form is submitted.
onmouseover	occurs when mouse is moved over an element.
onmouseout	occurs when mouse is moved out from an element (after moved over).
onmousedown	occurs when mouse button is pressed over an element.
onmouseup	occurs when mouse is released from an element (after mouse is pressed).
onload	occurs when document, object or frameset is loaded.
onunload	occurs when body or frameset is unloaded.
onscroll	occurs when document is scrolled.
onresized	occurs when document is resized.
onreset	occurs when form is reset.
onkeydown	occurs when key is being pressed.
onkeypress	occurs when user presses the key.
onkeyup	occurs when key is released.

# window.open()

Opens the new window.

*syntax*

```
var window = window.open(url, windowName, [windowFeatures]);
```

A DOM String indicating the URL of the resource to be loaded. This can be a path or URL to an HTML page, image file, or any other resource which is supported by the browser. If the empty string ("") is specified as url, a blank page is opened into the targeted browsing context.

```
<script>
    function openwin(){
        mywin=window.open("Page1.html","","","width=200,height=200,top=200,left=100");
    }
</script>
```

# window.close()

closes the current window

*syntax*

`window.close();`

```
<script>
    window.close();
</script>
```

# window.print()

prints the contents of the window.

*syntax*

```
window.print();
```

```
<script>
    window.print();
</script>
```

# JavaScript Global Properties & methods

**Infinity:** A numeric value that represents positive/negative infinity

**NaN:** "Not-a-Number" value

**eval():** Evaluates a string and executes it as if it was script code

**isFinite():** Determines whether a value is a finite, legal number

**isNaN():** Determines whether a value is an illegal number

**Number():** Converts an object's value to a number

**parseFloat():** Parses a string and returns a floating point number

**parseInt():** Parses a string and returns an integer

**END**

# Session-2

# Topics to be covered...

- JavaScript Array
- JavaScript Array Methods
- JavaScript String
- String Methods
- JavaScript Dates Get/Set Methods
- JavaScript Math Object
- JavaScript RegEx

# Array

JavaScript array is an object that represents a collection of similar type of elements.

Array indexes start with 0. [0] is the first array element, [1] is the second, [2] is the third ...

By array literal:

```
var arrayname=[“value1”,“value2”.....“valueN”];
```

By using an Array constructor (using new keyword):

```
var arrayname=new Array(“value1”,“value2”,“value3”);
```

```
<script>
var emp=new Array("one","two","three");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

**Output:**

one  
two  
three

# Array Properties

**length property:** The `length` property of an object which is an instance of type Array sets or returns the number of elements in that array.

Syntax: `array.length`

```
<script type="text/javascript">
var arr = new Array( 10, 20, 30 );
document.write("array length is : " + arr.length);
</script>
```

# Array-methods

**concat()**: Returns a new array comprised of this array joined with other array(s) and/or value(s).

Syntax:

```
array.concat(value1, value2, ..., valueN);
```

**toString()**: Returns a string representing the array and its elements.

Syntax:

```
array.toString();
```

**join()**: Joins all elements of an array into a string.

It behaves just like `toString()`, but in addition we can specify the separator.

Syntax:

```
array.join(separator);
```

**pop():**

Removes the last element from an array and returns that element.

Syntax:

```
array.pop();
```

**push():**

Adds one or more elements to the end of an array and returns the new length of the array.

Syntax:

```
array.push(element1, ..., elementN);
```

**shift():** Removes the first element from an array and returns that element.

Syntax:

```
array.shift();
```

**unshift():** Adds one or more elements to the front of an array and returns the new length of the array.

Syntax:

```
array.unshift( element1, ..., elementN );
```

**splice()**: Adds and/or removes elements from an array.

Syntax:

array.splice(index, howMany, [element1][, ..., elementN]);

**index** – Index at which to start changing the array.

**howMany** – An integer indicating the number of old array elements to remove.

If **howMany** is 0, no elements are removed.

**element1, ..., elementN** – The elements to add to the array. If you don't specify any elements, splice simply removes the elements from the array.

**slice()**: Extracts a section of an array and returns a new array.

Syntax:

array.slice( begin [,end] );

**It does not remove any elements from the source array.**

## sort() Sorts the elements of an array

Syntax:

array.sort( );

array.sort(compareFunction);

When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (*negative, zero, positive*) value

```
<script type="text/javascript">
    var num = [10,20,20,10];
    num.sort(function (a , b) {
        console.log(a + " & " + b + " " + (a-b));
    });
</script>
```

**indexOf()**: Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.

Syntax:

```
array.indexOf(searchElement[, fromIndex]);
```

**lastIndexOf()**: Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.

Syntax:

```
array.lastIndexOf(searchElement[, fromIndex]);
```

**reverse()**: Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.

Syntax:

```
array.reverse();
```

## **Array.forEach()**

The forEach() method calls a function (a callback function) once for each array element.

## **Array.map()**

The map() method creates a new array by performing a function on each array element.

The map() method does not execute the function for array elements without values.

The map() method does not change the original array.

## **Array.filter()**

The filter() method creates a new array with array elements that passes a test.

## **Array.reduce()**

The reduce() method reduces the array to a single value.

# String Object

The String object is a constructor for strings, or a sequence of characters.

Syntax:

```
var s = new String(string);
```

**Length:** Returns the length of the string.

## Special Characters:

Code	Result	Description
\'	'	Single quote
\\"	"	Double quote
\\\	\	Backslash

# String Methods

**charAt()**:Returns the character at the specified index

Syntax:

`string.charAt(index);`

**index** – An integer between 0 and 1 less than the length of the string.

Return Value

Returns the character from the specified index.

**charCodeAt()**Returns a number indicating the Unicode value of the character at the given index.

Syntax:

`string.charCodeAt(index);`

**match()**:Used to match a regular expression against a string.

Syntax:

`string.match( param )`

**param** – A regular expression object.

**replace()** Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.

Syntax:

```
string.replace(regexp/substr, newSubStr/function[, flags]);
```

```
<script>  
var re = "apples";  
  
var str = "Apples are round, and apples are juicy.";  
  
var newstr = str.replace(re, "oranges");  
  
document.write(newstr );  
</script>
```

To replace case insensitive, use a **regular expression** with an **/i** flag (insensitive):

To replace all matches, use a **regular expression** with a **/g** flag (global match)

**search()** Executes the search for a match between a regular expression and a specified string.

Syntax:

```
string.search(regexp);
```

**Str.toLowerCase()**: Returns the calling string value converted to lower case.

**Str.toUpperCase()**: Returns the calling string value converted to uppercase.

**endsWith()**: Checks whether a string ends with specified string/characters

**split()**Splits a String object into an array of strings by separating the string into substrings.

Syntax:

string.split([separator][, limit]);

```
var x1 = ("Welcome to infoway technologies pune ".split());  
var x2 = (" Welcome to infoway technologies pune ".split(" "));  
var x3= (" Welcome to infoway technologies pune ".split(" ", 2));
```

**substr()**Returns the characters in a string beginning at the specified location through the specified number of characters.

Syntax:

string.substr(start[, length]);

```
document.write( str.substr(-2,2));
```

```
document.write(str.substr(1));
```

```
document.write( str.substr(-20,2));
```

**substring()** Returns the characters in a string between two indexes into the string.

Syntax:

string.substring(indexA, [indexB])

**localeCompare():** Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.

Syntax:

string.localeCompare( param )

**0** – If the string matches 100%.

**1** – no match, and the parameter value comes before the *string* object's value in the locale sort order

**-1** – no match, and the parameter value comes after the *string* object's value in the local sort order

# Math object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Math.round(x) returns the value of x rounded to its nearest integer

Math.pow(x, y) returns the value of x to the power of y

Math.sqrt(x) returns the square root of x

Math.abs(x) returns the absolute (positive) value of x

Math.ceil(x) returns the value of x rounded **up** to its nearest integer

Math.floor(x) returns the value of x rounded **down** to its nearest integer

Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments

**Math.random()** returns a random number between 0 (inclusive), and 1 (exclusive).

```
Math.floor(Math.random() * 10);      // returns a number between 0  
                                         and 9
```

```
Math.floor(Math.random() * 11);      // returns a number between 0  
                                         and 10
```

```
Math.floor(Math.random() * 100);     // returns a number between 0  
                                         and 99
```

```
Math.floor(Math.random() * 10) + 1;  // returns a number between 1  
                                         and 10
```

```
Math.floor(Math.random() * 100) + 1; // returns a number between 1  
                                         and 100
```

# Date Object

A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.

Date objects are created with the **new Date()** constructor.

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

```
<script>
var d = new Date();
document.write(d.getDay());
</script>
```

## Date set Methods:

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

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

# Regular Expressions

## Regular Expression Modifiers

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

## Quantifiers

Quantifier	Description
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n

**\$ : Matches character at the end of the line.**

**.: Matches only period.**

**^ : Matches the beginning of a line or string.**

**- : Range indicator. [a-z, A-Z]**

[0-9] : Find any character between the brackets (any digit)

[^0-9] : Find any character NOT between the brackets (any non-digit)

[abc] : Find any character between the brackets

[^abc] : Find any character NOT between the brackets

(x|y)Find any of the alternatives specified

[a-z] : It matches characters from lowercase ‘a’ to lowercase ‘z’.

[A-Z] : It matches characters from uppercase ‘A’ to lowercase ‘Z’.

\w: Matches a word character and underscore. (a-z, A-Z, 0-9, \_).

\W: Matches a non word character (%, #, @, !).

\d: Find a digit

\s: Find a whitespace character

{M, N} : Denotes minimum M and maximum N value.

## RegEx Object Methods:

exec(): Tests for a match in a string. Returns the first match

test(): Tests for a match in a string. Returns true or false

**END**

# Session-3 & 4

# Topics to be covered...

- DOM Methods
- DOM Document
- DOM Elements
- DOM HTML
- DOM CSS
- DOM Animations
- DOM Event Listener
- JavaScript Form Validation
- DOM Nodes

# *HTML DOM*

The **document object** represents the whole html document.

When a webpage loads in the browser. browser creates a DOM for the page.

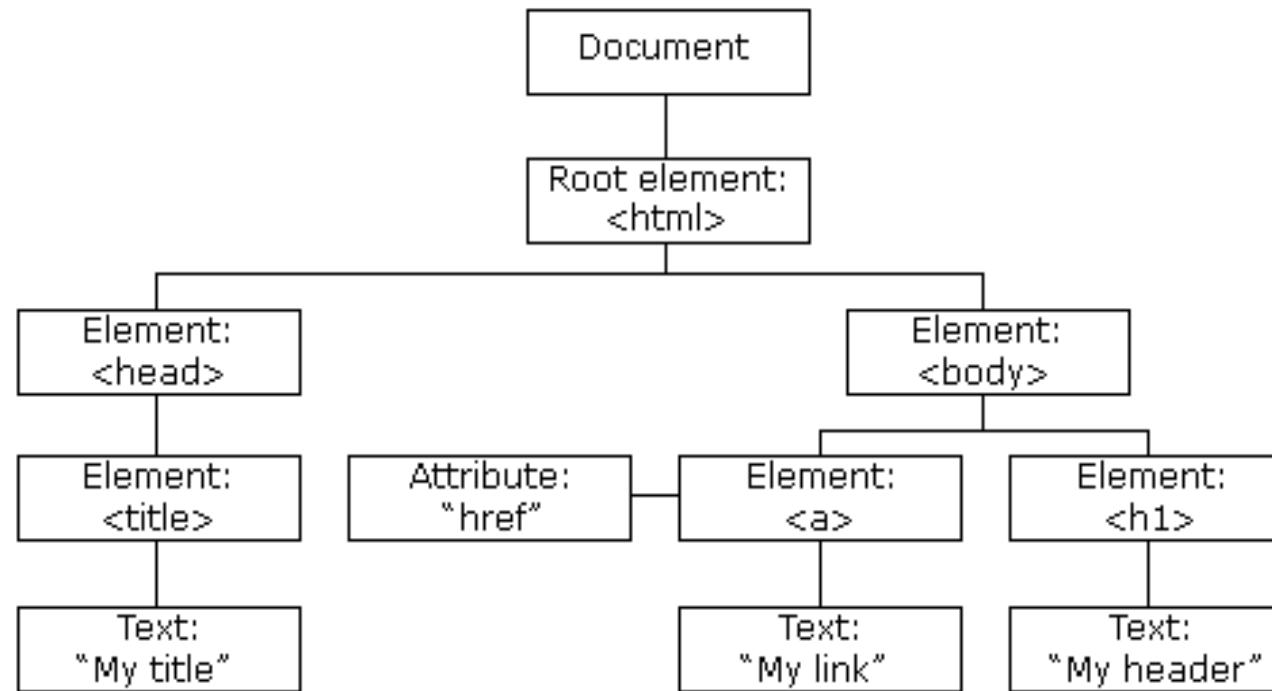
**Document:- Html Page**

**Object:- Elements and attributes**

**Model:- Tree Structure of HTML elements**

By the help of document object, we can add dynamic content to our web page.

The DOM is a W3C (World Wide Web Consortium) standard.



**Why we should learn HTML DOM:** w3 standard to understand the structure of html page so that we can create ,read, update and delete and manage DOM elements using JavaScript and jQuery methods.

HTML DOM helps you to understand and control the element structure using JavaScript methods.

**document.write("string")**

writes the given string on the document.

**document.getElementById()**

returns the element having the given id value.

**getElementsByName()**

returns all the elements having the given name value.

**getElementsByTagName()**

returns all the elements having the given tag name.

**getElementsByClassName()**

returns all the elements having the given class name

# innerHTML Property

The **innerHTML** property can be used to write the dynamic html on the html document.

Syntax:

```
document.getElementById(id).innerHTML = new HTML
```

```
<script>
document.getElementById("para").innerHTML = "New text!";
</script>

<p id="para">Old text!</p>
```

# innerText Property

The `innerText` property sets or returns the text content of the specified node.

Syntax:

```
document.getElementById(id).innerText = text
```

```
<script>
document.getElementById("para").innerText = "New text!";
</script>
```

```
<p id="para">Old text!</p>
```

# DOM createElement()

The createElement() method creates an Element Node with the specified name.

*syntax*

```
var element = document.createElement(tagName);
```

```
var para= document.createElement("p");
```

# DOM createTextNode()

To add text to the element we have to create a textnode first.

*syntax*

```
var node= document.createTextNode("text");
```

```
var node = document.createTextNode("This is a new  
paragraph.");
```

# HTML DOM textContent Property

The `textContent` property sets or returns the textual content of the specified node.

*syntax*

`node.textContent = text`

```
Para.textContent="This is a new paragraph.;"
```

# HTML DOM setAttribute() Method

The `setAttribute()` method adds the specified attribute to an element, and gives it the specified value.

*syntax*

`element.setAttribute(attributeName, attributeValue)`

# DOM Node.appendChild()

The `Node.appendChild()` method adds a node to the end of the list of children of a specified parent node.

*syntax*

`Node.appendChild()`

```
<div id="div1">
</div>

<script>
var para = document.createElement("p");
var node =
document.createTextNode("Infoway");
para.appendChild(node);

var element =
document.getElementById("div1");
element.appendChild(para);
</script>
```

# HTML DOM EventListener

The addEventListener() method attaches an event handler to the specified element.

You can add many event handlers to one element

*Syntax:*

`element.addEventListener(event, function);`

```
<button id="myBtn">Handler</button>

<script>
document.getElementById("myBtn").addEventListener("click",
myFunction);

function myFunction() {
    alert ("Hello World!");
}
</script>
```

# Event Bubbling or Event Capturing?

Event propagation is a way of defining the element order when an event occurs. If you have a `<p>` element inside a `<div>` element, and the user clicks on the `<p>` element, which element's "click" event should be handled first?

In *bubbling* the inner most element's event is handled first and then the outer: the `<p>` element's click event is handled first, then the `<div>` element's click event.

In *capturing* the outer most element's event is handled first and then the inner: the `<div>` element's click event will be handled first, then the `<p>` element's click event.

*Syntax:*

`addEventListener(event, function, useCapture)`

default value is `false` which will use the bubbling propagation,

**END**

# Session-5

# Topics to be covered...

- Canvas Methods
- JavaScript Window
- JavaScript Screen
- JavaScript Location
- JavaScript History
- JavaScript Navigator
- JavaScript Timing
  - setTimeout()
  - setInterval()
- JavaScript Objects
- Properties and Methods

- a. The HTML <canvas> element is used to draw graphics on a web page.
- b. Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

Step 1: Find the Canvas Element

```
var canvas = document.getElementById("myCanvas");
```

Step 2: Create a Drawing Object

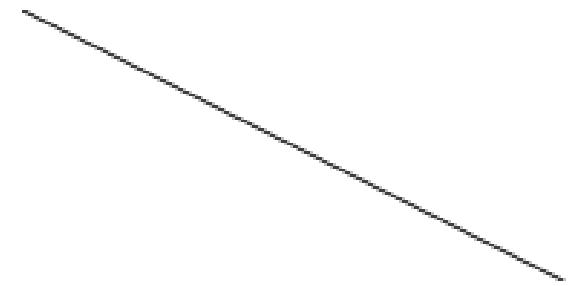
```
var ctx = canvas.getContext("2d");
```

## Draw a Line:

`moveTo(x,y)` - defines the starting point of the line

`lineTo(x,y)` - defines the ending point of the line

```
var canvas = document.getElementById("CN");
var ctx = canvas.getContext("2d");
ctx.moveTo(0, 0);
ctx.lineTo(200, 100);
ctx.stroke();
```

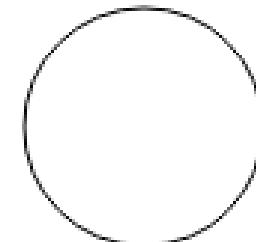


## Draw a Circle:

`beginPath()` - begins a path

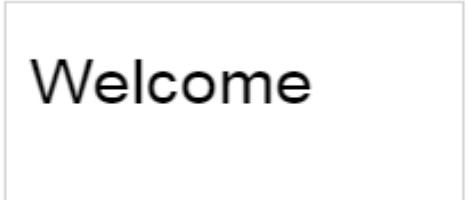
`arc(x,y,r,startangle,endangle)` - creates an arc/curve.

```
var canvas = document.getElementById("CN");
var ctx = canvas.getContext("2d");
ctx.beginPath();
ctx.arc(95, 50, 10, 0, 2 * Math.PI);
ctx.stroke();
```



## Drawing Text on the Canvas:

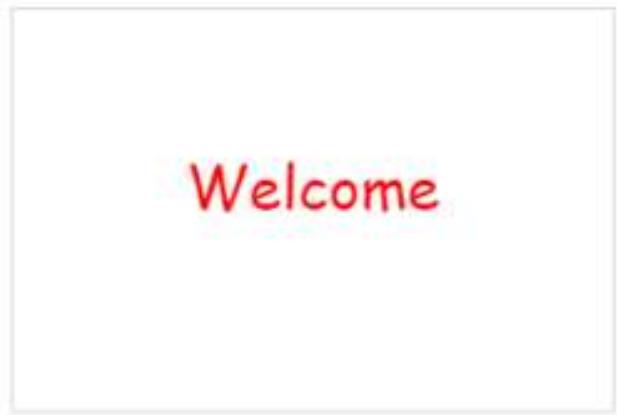
```
var canvas = document.getElementById("CN");
var ctx = canvas.getContext("2d");
ctx.font = "30px Arial";
ctx.fillText("Welcome", 10, 50);
```



Welcome

## Add Color and Center Text:

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.font = "30px Comic Sans MS";
ctx.fillStyle = "red";
ctx.textAlign = "center";
ctx.fillText("Welcome", 50,50);
```



Welcome

# Window location object

The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page.

**window.location.href** returns the href (URL) of the current page

**window.location.hostname** returns the domain name of the web host

**window.location.pathname** returns the path and filename of the current page

**window.location.protocol** returns the web protocol used (http: or https:)

**window.location.port** property returns the number of the internet host port (of the current page).

# Window History object

The window.history object contains the browsers history.

history.back() - same as clicking back in the browser

history.forward() - same as clicking forward in the browser

history.go() - loads the given page number.

```
history.back();      //for previous page  
history.forward(); //for next page  
history.go(2);     //for next 2nd page  
history.go(-2);    //for previous 2nd page
```

# Window navigator object

The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName etc.

<b>appName</b>	returns the name
<b>appVersion</b>	returns the version
<b>appCodeName</b>	returns the code name
<b>cookieEnabled</b>	returns true if cookie is enabled otherwise false
<b>language</b>	returns the language
<b>platform</b>	returns the platform e.g. Win32.
<b>online</b>	returns true if browser is online otherwise false..

## Window.setTimeout() Method:

Executes a function, after waiting a specified number of milliseconds.

Syntax:

`setTimeout(function, milliseconds);`

The first parameter is a function to be executed.

The second parameter indicates the number of milliseconds before execution.

```
<button onclick="setTimeout(myFunction,  
3000)">SetTimeout</button>
```

```
<script>  
function myFunction() {  
    alert('Hello');  
}  
</script>
```

## Window.setInterval() Method

The setInterval() method repeats a given function at every given time-interval.

Syntax:

```
setInterval(function, milliseconds);
```

The first parameter is the function to be executed.

The second parameter indicates the length of the time-interval between each execution.

```
<script>
var myVar = setInterval(starttime, 1000);

function starttime() {
    var d = new Date();
    document.write(d.toLocaleTimeString());
}
</script>
```

## clearInterval() method:

The clearInterval() method stops the executions of the function specified in the setInterval() method.

Syntax:

```
window.clearInterval(timerVariable)
```

```
<p id="starttime"></p>
```

```
<button onclick="clearInterval(stime)">Stop time</button>
```

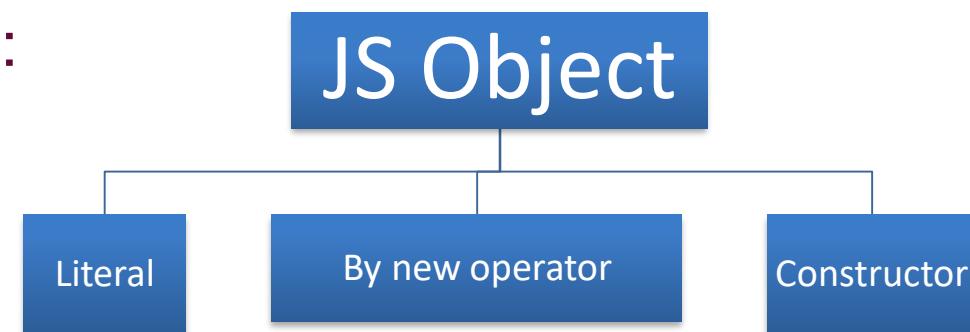
```
<script>
var stime= setInterval(stopWatch, 1000);
function stopWatch() {
    var d = new Date();
    document.getElementById("starttime").innerHTML =
d.toLocaleTimeString();
}
</script>
```

# Javascript Object

A javaScript object is an entity having state and behavior (properties and method).

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

Creating Objects in JavaScript:



# JavaScript Object by object literal

Syntax:

```
object={property1:value1,  
        property2:value2.....propertyN:valueN}
```

```
Var employee=  
{  
    name:"Ram Kumar",  
    salary:40000  
}  
document.write(employee.name+" "+employee.salary);
```

# method in JavaScript Object literal

```
<p id="demo"></p>

<script>
var person = {
    firstName: "Sohan",
    lastName : "Lal",
};
person.name = function() {
    return this.firstName + " " + this.lastName;
};
document.getElementById("demo").innerHTML ="Name is " +
person.name();
</script>
```

# For...in loop

Syntax:

```
for (variable in object) {  
    code to be executed  
}
```

The block of code inside of the for...in loop will be executed once for each property.

```
<script>  
var p;  
var person = {fname:"Ram", lname:"Sharma", age:25};  
  
for (x in person) {  
    p += person[x];  
}  
Document.write(p);  
</script>
```

# By The new Operator

All user-defined objects and built-in objects are descendants of an object called **Object**.

Syntax:

```
var objectname=new Object();
```

```
var employee=new Object();
employee.id=101;
employee.name="Ram Sharma";
employee.salary=50000;
document.write(employee.id+" "+employee.name+" "+employee.salary);
```

# Defining method

```
<script>
    var Person = new Object();
    Person.id = 1001;
    Person['n'] = 'Infoway Technologies';
    Person.getName = function () {
        return (Person.n);
    }
    alert(Person.getName());
</script>
```

# By using an Object constructor

We need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

```
function employee(){  
    this.name="Smith";  
    this.salary=50000;  
}
```

```
var e=new employee();  
Document.write(e.name+”  
“+e.salary);
```

```
<p id="demo"></p>
<script>
// Constructor function for Person objects
function Person(first, last, age) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.name = function() {
        return this.firstName + " " + this.lastName
    };
}

// Create a Person object
var p= new Person("Sohan", "Lal", 50);

// Display full name
document.getElementById("demo").innerHTML = p.name();

</script>
```

# Object Properties

An **object** is a collection of properties, and a property is an association between a *name* (or *key*) and a *value*. A property's value can be a **function**, in which case the property is known as a **method**.

*syntax*

- `objectName.property`      *// person.age*
- `objectName["property"]`      *// person["age"]*

```
<script type="text/javascript">
    var Person = new Object();
    Person.id = 1001;
    Person['n'] = 'Infoway Technologies';
    alert(Person.n);
</script>
```

# Deleting Properties

The **delete** keyword deletes a property from an object.

The delete keyword deletes both the value of the property and the property itself.

It has no effect on variables or functions.

The delete operator should not be used on predefined JavaScript object properties.

```
<script type="text/javascript">
    var Person = new Object();
    Person.id = 1001;
    delete Person.id;
    console.log(Person.id);
</script>
```

# JavaScript Object Prototypes

All JavaScript objects inherit properties and methods from a prototype.

## **Prototype Inheritance:**

The Object.prototype is on the top of the prototype inheritance chain.

Date objects, Array objects, and Person objects inherit from Object.prototype.

# Adding Properties and Methods to Objects

```
Person.prototype.course = "PG-DAC";
```

```
Person.prototype.name = function() {  
    return this.fname + " " + this.lname;  
};  
document.write(p.name());
```

# Function Expressions

A JavaScript function can also be defined using an **expression**.

```
var x = function (a, b) {return a * b}; //anonymous function (a function without a name).  
var z = x(4, 3);
```

function flyToTheMoon()  
{  
 alert("Zoom! Zoom! Zoom!");  
}  
flyToTheMoon();

function name  
function declaration

This diagram shows a standard function declaration. It highlights the word 'function' in blue and 'flyToTheMoon' in green. Red arrows point from these words to their respective labels: 'function name' and 'function declaration'. The code itself is in grey.

var flyToTheMoon = function()  
{  
 alert("Zoom! Zoom! Zoom!");  
}

variable that stores returned object  
function operator  
function body

This diagram shows a function expression. It highlights 'var' in blue, 'flyToTheMoon' in green, and the opening brace of the function in blue. Red arrows point from these words to their respective labels: 'variable that stores returned object', 'function operator', and 'function body'. The code is in grey.

## Function Hoisting:

```
myFunction(5);
```

```
function myFunction(y) {  
    return y * y;  
}
```

Functions defined using an expression are not hoisted

# Self-Invoking Functions

A self-invoking (also called self-executing) function is a nameless (anonymous) function that is invoked immediately after its definition.

An anonymous function is enclosed inside a set of parentheses followed by another set of parentheses (), which does the execution.

```
(function() {  
    ...  
})();
```

Self-invoking functions are useful for initialization tasks and for one-time code executions, without the need of creating global variables.

```
(function(){  
    console.log(Math.PI);  
})();
```

```
(function(x){  
    console.log(x);  
})("Hello, World!");
```

# Function Closure

Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.

Inner function can access variables and parameters of an outer function (however, cannot access arguments object of outer function).

```
var incrementClickCount = (function ()  
{  
    var clickCount = 0;  
    return function ()  
    {  
        return ++clickCount;  
    }  
}());  
  
<input type="button" value="Click Me" onclick="alert(incrementClickCount());" />
```

## When to use Closure?

Closure is useful in hiding implementation detail in JavaScript. In other words, it can be useful to create private variables or functions.

# Cookie

A little “tarball” of information stored on the client machine’s hard drive.

Usually in the cookies.txt file

information can be placed in the cookies.txt file two ways:

- by the client using Javascript
- by the server placing the cookie in the HTTP Response Header
  - in this case, you’ll usually be asked if it’s OK to save the cookie by your browser

The “tarball” consists of name value pairs

Cookies must be no larger than 4 KB

# Setting cookies

```
document.cookie = "cookieName = cookieData  
[ ; expires = timeInGMTString]  
[ ; path = pathName ]  
[ ; domain = domainName ]  
[ ; secure ]"
```

- Name/data - each cookie must have a name and data; data can contain no space
- expires - must be in GMT format (use toGMTString function); if expires is omitted cookie is temporary and won't be written to cookies.txt
- path - for client side cookies just use the default
- Domain - to help synchronize cookie data across a set of documents for the specified domain (this is known as the cookie domain)
- SECURE - this is really for server-side cookies, so omit it.

# Cookies

## Storage typically used prior to HTML5

- Before HTML5, websites could store only small amounts of text-based information on a user's computer using cookies.
- A website might use a cookie to record user preferences or other information that it can retrieve during the client's subsequent visits.
- When a user visits a website, the browser locates any cookies written by that website and sends them to the server.
- Cookies may be accessed only by the web server and scripts of the website from which the cookies originated. The browser sends these cookies with every request to the server.
- Cookies have a number of problems associated with them.
  - Offline access is not often performed very well.
  - memory is limited.

# HTML5 Storage

HTML web storage provides two objects for storing data on the client:

`window.localStorage` - stores data with no expiration date

`window.sessionStorage` - stores data for one session (data is lost when the browser tab is closed)

```
<body>
    <input type="text" id="name"><input type="button" id="btn" value="Set Storage"><br>
    <input type="button" id="btn1" value="Get Storage"><br>
    <div id="data"></div>
    <script>
        document.getElementById('btn').onclick=function(){
            var n=document.getElementById('name').value;
            localStorage.setItem("Username",n);
        }
        document.getElementById('btn1').onclick=function(){
            document.getElementById("data").innerHTML=localStorage.Username
            // sessionStorage.getItem("Username")
        }
    </script>
</body>
```

**END**

# Thank you

# **Web Programming Technologies**

Harshita Maheshwari

# Session-8



# Topics to be covered...

- Introduction to jQuery
- Why we use jQuery
- Ready function
- jQuery Selectors
- jQuery HTML
- jQuery Effects
- jQuery Events
- jQuery Form Validation
- jQuery UI

# Introduction to jQuery

jQuery is a lightweight JavaScript library that emphasizes interaction between JavaScript and HTML

It is cross-platform and supports different types of browsers

Developed in 2006 by John Resig at Rochester Institute of Technology

## jQuery

Select  
elements  
by  
tag, class  
, Id

Handle  
events in  
same way  
across  
browsers

Supports  
for  
animation

Eases  
service  
and Ajax  
calls

helps in  
applying  
styles to  
multiple  
elements

# Why jQuery

jQuery is like a pre-packaged set of JavaScript routines that you may have otherwise needed to write yourself, packaged in an easy-to-use way.

It uses CSS syntax for selection of element.

jQuery makes animated applications just like Flash

jQuery pages load faster

# jQuery vs. JavaScript

- Example 1 - Hide an element with id "textbox"

//javascript

```
document.getElementById('textbox').style.display = "none";
```

//jQuery

```
$('#textbox').hide();
```

- Example 2 - Create a <h1> tag with "my text"

//javascript

```
var h1 = document.createElement("h1");
h1.innerHTML = "my text";
document.getElementsByTagName('body')[0].appendChild(h1);
```

//jQuery

```
 $('body').append( $("<h1>").html("my text") );
```

# Getting started with jQuery

Two ways to access it:

-Download it and reference:

```
<script src="jquery.js"></script>  
(in the same directory)
```

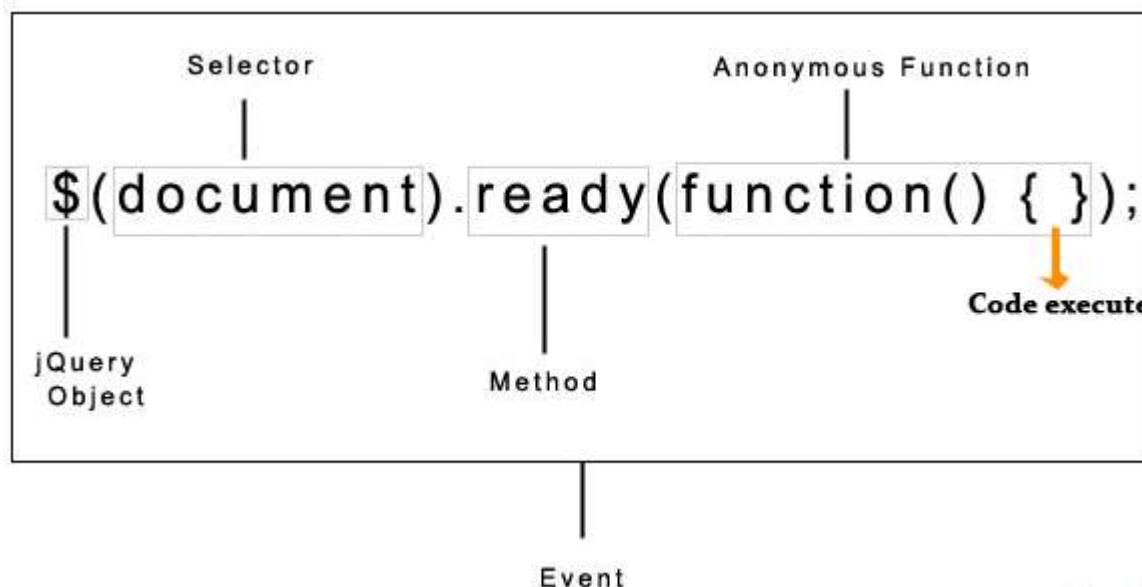
-Reference the JS file in your HTML (CDN):

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.js">  
</script>
```

# `$( document ).ready()`

A page can't be manipulated safely until the document is "ready."

jQuery detects this state of readiness for you. Code included inside `$( document ).ready()` will only run once the page Document Object Model is ready for JavaScript code to execute.



jQuery selectors allow you to select and manipulate HTML element(s).

- TagName

```
document.getElementsByTagName("tagName");  
$("tagName") - $("div"), $("p"), $("div"),.....
```

- Tag ID

```
document.getElementById("id");  
$("#id") - $("#name"), $("#address")
```

- Tag Class

```
document.getElementsByClassName("className");  
$(".className") - $(".comment"), $(".code")
```

- To select all elements - \$("\*")

`$("tagName,.className")`

**Example:** `$("h1,.mainTitle")`

`$("tagName,.className,#tagId")`

**Example:** `$("h1,.mainTitle,#firstHeading")`

## Examples:

`$("selector:first")`

`$("p:first")`

`$("selector:last")`

`$("p:last")`

`$("selector:odd")`

`$("p:odd")`

`$("selector:even")`

`$("p:even")`

`$("selector[attribute]")`

`$("p:[name]")`

# Condition filters - Form filters

`$("selector:visible")`

`$("selector:input")`

`$("selector:disabled")`

`$("selector:text")`

`$("selector:enabled")`

`$("selector:password")`

`$("selector:checked")`

`$("selector:checkbox")`

`$("selector:button")`

`$("selector:submit")`

`$("selector:file")`

`$("selector:reset")`

# :checkbox selector

```
<script>
$(document).ready(function(){
$("button").click(function(){
  $("input:checkbox").each(function(k,v){
    alert(k+v+v.value);
  });
}); });
</script>
```

```
$(document).ready(function(){
  $(":checkbox").click(function () {
    var x = "";
    $("#span1").text("");
    $(":checked").each(function (k, v) {
      x += v.value;
    });
    $("#span1").text(x);
  })
});
</script>
```

# Retrieve, Set and Remove attributes

## Examples:

```
$(“selector”).attr(“name”)
```

```
$(“img”).attr(“src”)
```

```
$(“selector”).attr(“key”, “val”)
```

```
$(“p”).attr(“class”, “source”)
```

```
$(“selector”).attr(properties)
```

```
$(“img”).attr({ “src” : “/path/”,
```

```
“title” : “My Img” });
```

```
$(“selector”).removeAttr(attr)
```

```
$(“div”).removeAttr(“class” )
```

```
$(“selector”).addClass(“className”)
$(“selector”).removeClass(“className”)
$(“selector”).toggleClass(“className”)
```

**html()** - Sets or returns the content of selected elements (including HTML markup)

```
$(“selector”).html()
$(“selector”).html(“html code”)
```

**text()** - Sets or returns the text content of selected elements

```
$(“selector”).text()
$(“selector”).text(“text content”)
```

**val()** - Sets or returns the value of form fields

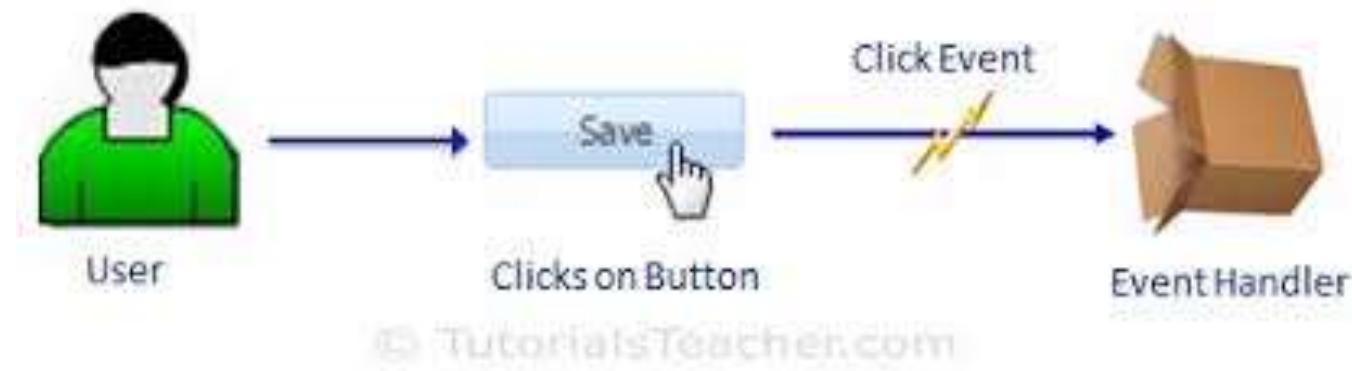
```
$(“selector”).val()
$(“selector”).val(“value”)
```

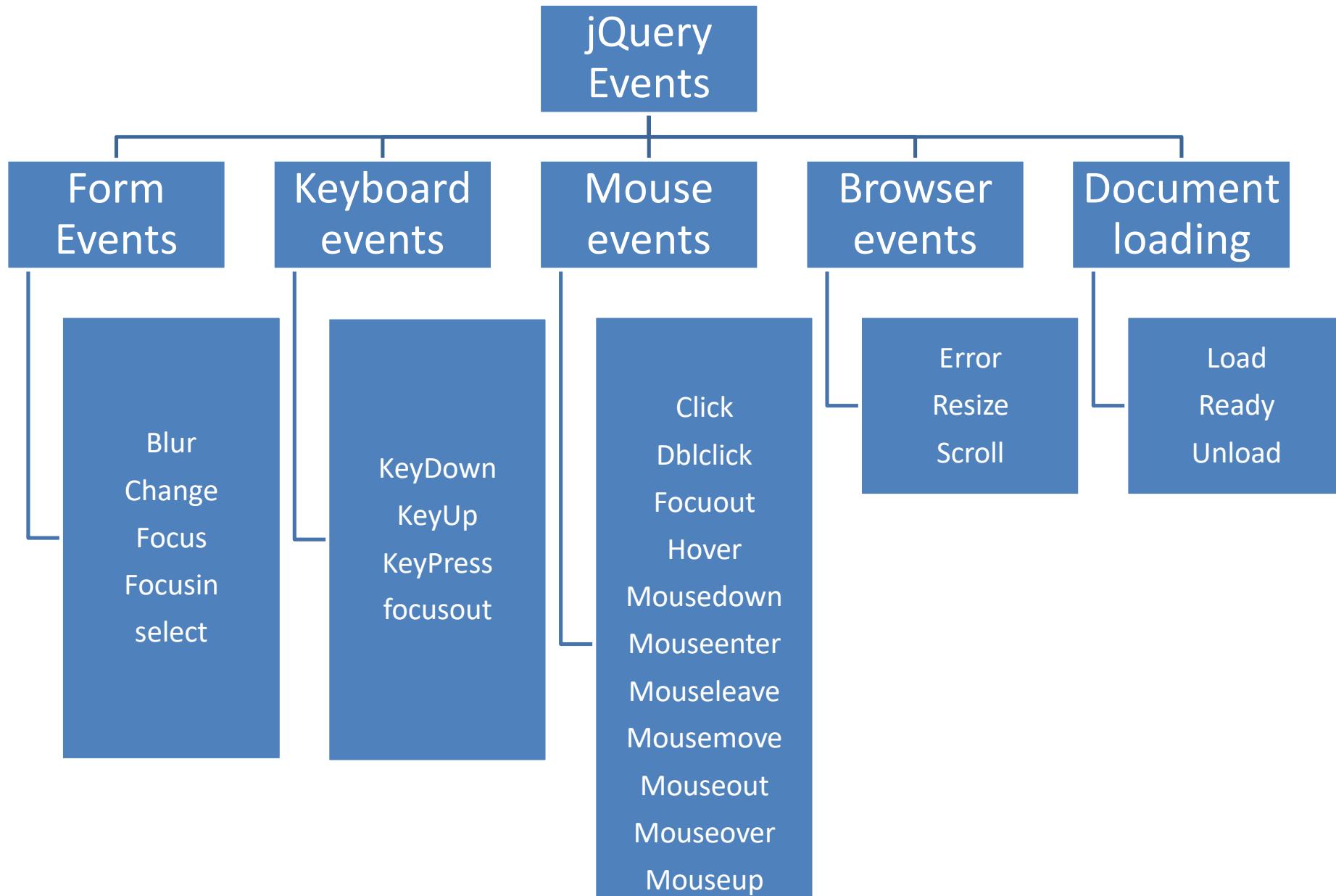
```
<script>
$(document).ready(function () {
    $("#button1").click(function () {
        alert($("#p1").text());
        alert($("#div1").html());
        alert($("#text1").val());
    });
});
</script>

<script>
$(document).ready(function () {
$("#btn1").click(function(){
    $("#p1").text("Hello world!");
});
$("#btn2").click(function(){
    $("#p2").html("<b>Welcome</b>");
});
$("#btn3").click(function(){
    $("#text1").val("Infoway");
});
});
</script>
```

# jQuery Events

Events are a part of the Document Object Model (DOM) and every HTML element contains a set of events which can trigger JavaScript Code.





```
// Event setup using a convenience method
$( "p" ).click(function() {
  console.log( "You clicked a paragraph!" );
});
```

```
// Equivalent event setup using the `on()` method
$("p" ).on( "click", function() {
  console.log( "You clicked a paragraph!" );
});
```

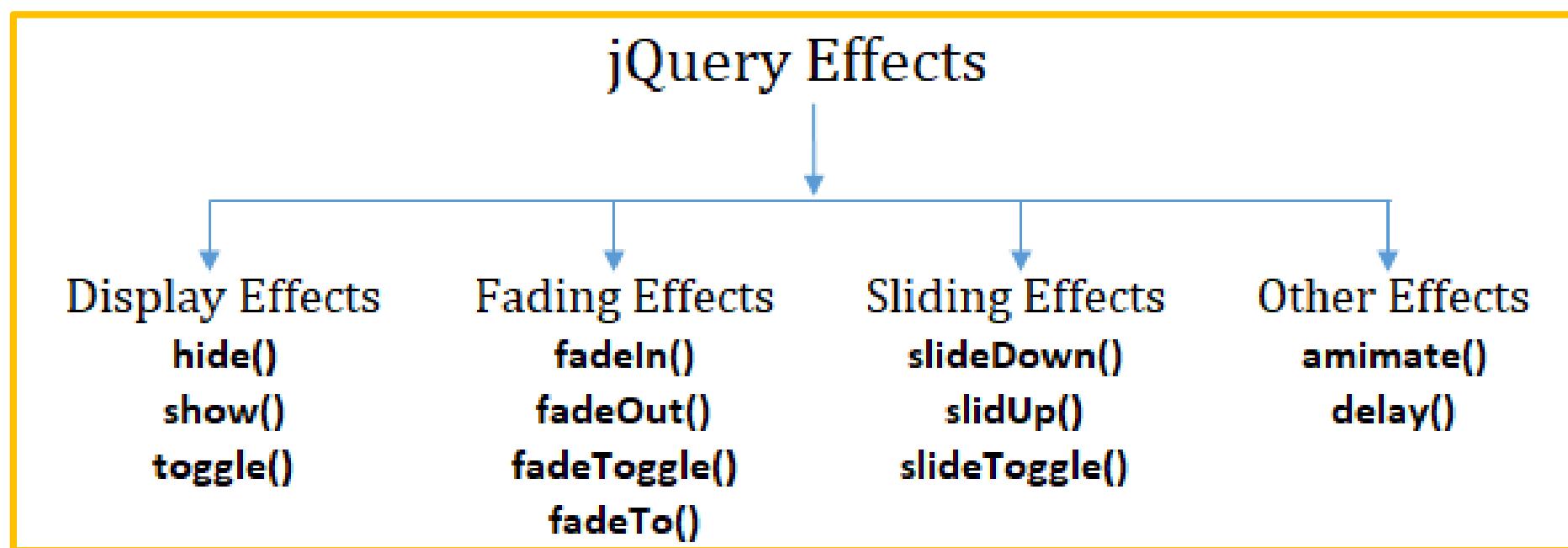
# Focusout event

The focusout event is sent to an element when it, or any element inside of it, loses focus. Bind an event handler to the "focusout" event.

```
<script>
  $(document).ready(function () {
    $(":text").focusin(function () {
      $(this).css("background-color", "red");
    });
    $(":text").focusout(function () {
      $(this).css("background-color", "white");
    });
  });
</script>
```

# jQuery Effects

The jQuery library provides several techniques for adding animation to a web page. These include simple, standard animations that are frequently used, and the ability to craft sophisticated custom effects.



Hide the matched elements.

Show the matched elements

**Syntax:**

`\$(selector).hide(speed,callback);`

`\$(selector).show(speed,callback);`

Speed is given in milliseconds; higher values indicate slower animations, not faster ones. The strings 'fast' and 'slow' can be supplied to indicate durations of 200 and 600 milliseconds, respectively.

**jQuery toggle():**

**Syntax:**

`\$(selector).toggle(speed,callback);`

# Fading Effects - .fadeIn(), .fadeOut()

Display the matched elements by fading them to opaque.

Hide the matched elements by fading them to transparent.

*syntax*

`$(selector).fadeIn(speed,callback);`

`$(selector).fadeOut(speed,callback);`

Speed is given in milliseconds; higher values indicate slower animations, not faster ones. The strings 'fast' and 'slow' can be supplied to indicate durations of 200 and 600 milliseconds, respectively. If any other string is supplied, or if the duration parameter is omitted, the default duration of 400 milliseconds is used.

# Fading Effects - .fadeTo(), .fadeToggle()

Adjust the opacity of the matched elements.

Display or hide the matched elements by animating their opacity.

Syntax:

```
$(selector).fadeTo(speed,opacity,callback);  
$(selector).fadeToggle(speed,callback);
```

opacity: A number between 0 and 1 denoting the target opacity.

```
<script>  
$(document).ready(function () {  
    $("#toggle").on("click", function () {  
        $("#div1").fadeTo("slow",0.5, function () {  
            alert("Hello World!");  
        });  
    });  
});  
</script>
```

# Sliding Effects - .slideDown(), .slideUp()

Display the matched elements with a sliding motion.

Hide the matched elements with a sliding motion.

Syntax:

`$(selector).slideDown(speed,callback);`

`$(selector).slideUp(speed,callback);`

`$(selector).slideToggle(speed,callback);`

# Stop()

The jQuery stop() method is used to stop animations or effects before it is finished.

Syntax:

```
$(selector).stop();
```

# Other Effects - .animate()

Perform a custom animation of a set of CSS properties.

Syntax:

`$(selector).animate({params},speed,callback);`

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!

HTML

```
<button id="left">left</button>
<button id="right">right</button>
<div class="block"></div>
```

JS

```
$(document).ready(function(){
  $(".block").css({
    'position': 'absolute',
    'backgroundColor': "#abc",
    'left': '100px',
    'width': '90px',
    'height': '90px',
    'margin': '5px' });
  $("#left").click(function(){
    $(".block").animate({left: "-=50px"}, "slow");
  });
  $("#right").click(function(){
    $(".block").animate({left: "+=50px"}, "slow");
  });
});
```

These methods get and set CSS-related properties of elements.

Set a CSS Property:

Syntax:

```
css("propertyname", "value");
```

```
$( "p" ).css( "background-color" , "yellow" );
```

## jQuery - Add Elements

- `append()` - Inserts content at the end of the selected elements
- `prepend()` - Inserts content at the beginning of the selected elements
- `after()` - Inserts content after the selected elements
- `before()` - Inserts content before the selected elements

## jQuery - Remove Elements

- `remove()` - Removes the selected element (and its child elements)
- `empty()` - Removes the child elements from the selected element

A jQuery UI widget is a specialized jQuery plug-in. Using plug-in, we can apply behaviours to the elements.

## **Autocomplete**

Enable to provides the suggestions while you type into the field.

## **Menu**

Menu shows list of items.

## **Datepicker**

It is to open an interactive calendar in a small overlay.

## **Select menu**

Enable a style able select element/elements.

## **Tooltip**

Its provides the tips for the users.

## **Tabs**

It is used to swap between content that is broken into logical sections.

# Thank You

# ES6 Features

Harshita Maheshwari

# ECMA Script6

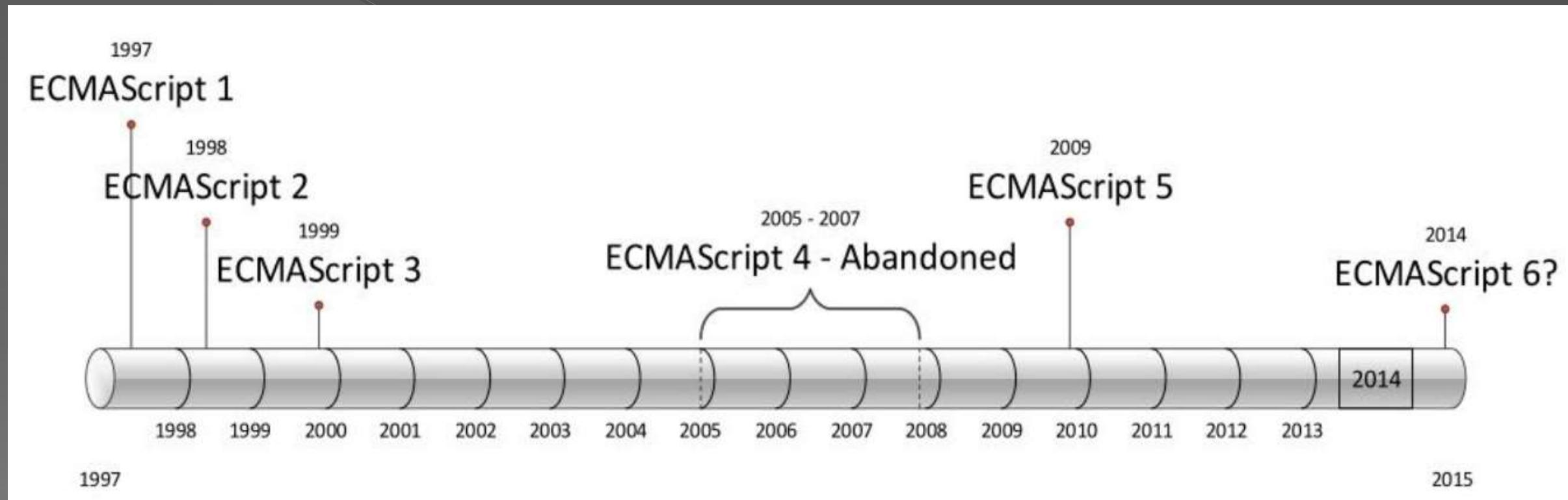


JavaScript ES6 (also known as ECMAScript 2015 or ECMAScript 6) is the newer version of JavaScript that was introduced in 2015.

ECMAScript is the standard that JavaScript programming language uses. ECMAScript provides the specification on how JavaScript programming language should work.

JavaScript ES6 brings new syntax and new awesome features to make your code more modern and more readable. It allows you to write less code and do more.

# ECMA Script History



# Features

1. let and const keywords
2. Arrow Functions
3. Multi-line Strings
4. Template Literals
5. Default Parameters
6. Rest Parameter and Spread Operators
7. Destructuring Assignment
8. for of loop
9. Classes
10. Inheritance
11. Modules



# let and const

<b>var</b>	<b>let</b>	<b>const</b>
Function scope	Block scope	Block scope
Hoisted	Not Hoisted	Not Hoisted
Can declare and initialization can be done later	Can declare and initialization can be done later	Must be initialized at the time of declaration
Can change	Can change	Read-only
Use for top level variables	Use for localized variable in smaller scope	Use for read-only/fixed values

## let

```
1 for (let i = 0; i < 10; ++i) {  
2   console.log(i); // 0, 1, 2, 3, 4 ... 9  
3 }  
4  
5 console.log(i); // i is not defined
```

## const

```
1 const PI = 3.14159265359;  
2 PI = 0; // => 0  
3 console.log(PI); // => 3.14159265359
```

```
function f() {  
  let x;  
  {  
    // okay, block scoped name  
    const x = "sneaky";  
    // error, const  
    x = "foo";  
  }  
  // error, already declared in block  
  let x = "inner";  
}
```

# Arrow Functions

Shorthand for Function Expression.  
Makes your code more readable, more structured, and look like modern code.

```
// ES5

function myFunc(name) {
    return 'Hello' + name;
}

console.log(myFunc('said'));

// output
// Hello said
```

```
// ES6 Arrow function

const myFunc= name =>{
    return `Hi ${name}`;
}
console.log(myFunc('Said'))// output Hi Said

// or even without using arrow or implement 'return' keyword
const myFunc= name => `Hi ${name}`;

console.log(myFunc('Said')) // output Hi Said
```

We can use Arrow function with map, filter, and reduce built-in functions.

```
// ES5

const myArray=['tony','Sara','Said',5];

let Arr1= myArray.map(function(item){
    return item;
});
console.log(Arr1); //output (4) ["tony", "Sara", "Said", 5]

//ES6 Syntax

let Arr2 = myArray.map(item => item);
console.log(Arr2); //output (4) ["tony", "Sara", "Said", 5]
```

# Multi-line Strings



We can create multi-line strings by using back-ticks(`).

It can be done as shown below :

```
let greeting = `Hello World,  
Greetings to all,  
Keep Learning and  
Practicing!`
```

# Template Literals

Syntactic sugar for string construction.

We don't have to use the plus (+) operator to concatenate strings, or when we want to use a variable inside a string.

```
//ES5

function myFunc1(name,age){
    return 'Hi' + name + ' Your age is' + age + 'year old!';
}
console.log(myFunc1('Said',22))
//output -->Hi Said,Your age is 22year old!
```

```
//ES6

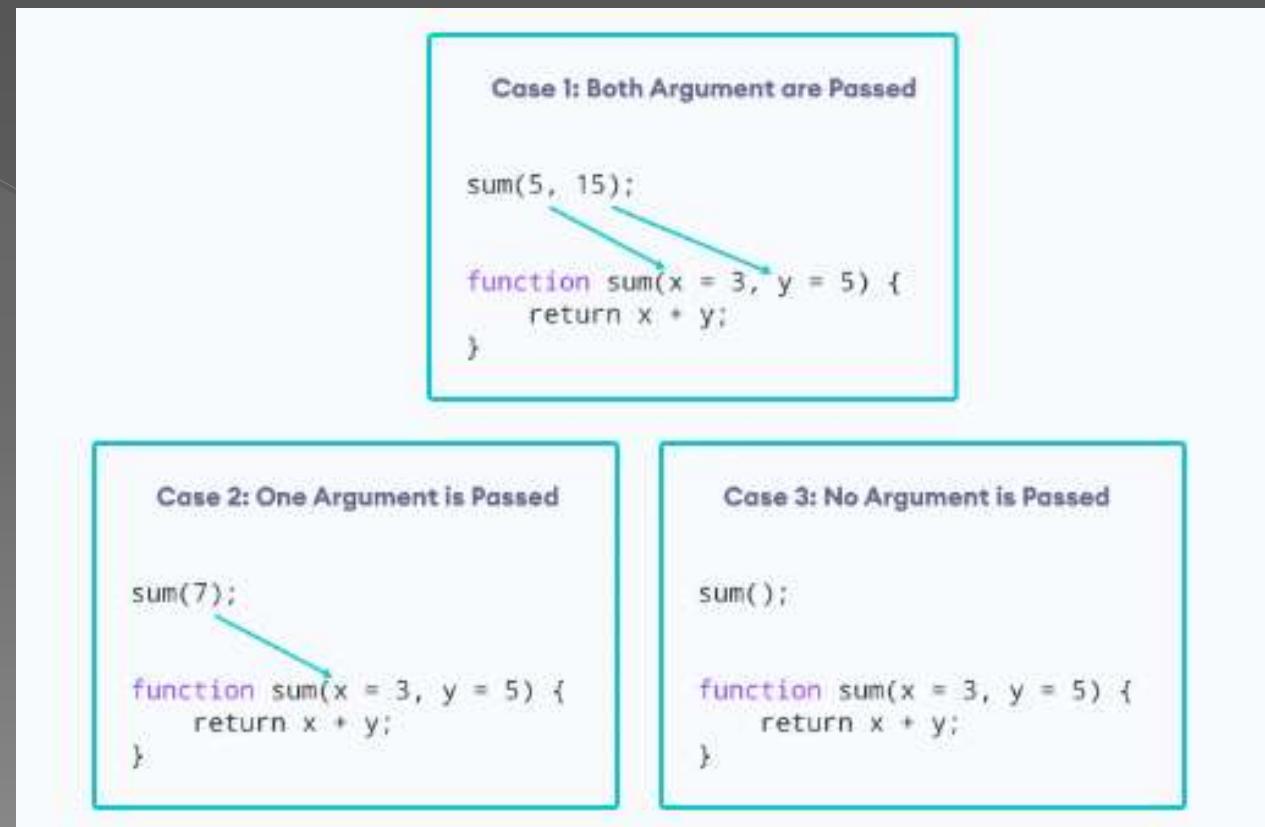
const myFunc= (name,age)=>{
    return `Hi ${name},Your age is ${age}year old!`;
}
console.log(myFunc('Said',22))
//output--> Hi Said,Your age is 22year old!
```

# Default Parameter

Allows us to give default values to function parameters.

We can provide the default values right in the signature of the functions.

```
function sum(x = 3, y = 5) {  
  
    // return sum  
    return x + y;  
}  
  
console.log(sum(5, 15)); // 20  
console.log(sum(7)); // 12  
console.log(sum()); // 8
```



# Rest parameter

The rest parameters allows a function to accept an indefinite number of arguments as an array.

```
function myFun(a, b, ...manyMoreArgs) {  
    console.log("a", a)  
    console.log("b", b)  
    console.log("manyMoreArgs", manyMoreArgs)  
}  
  
myFun("one", "two", "three", "four", "five", "six")  
  
// Console Output:  
// a, one  
// b, two  
// manyMoreArgs, ["three", "four", "five", "six"]
```

# Spread Operator

The spread operator ... is used to expand or spread an iterable or an array.

```
function sum(x, y, z) {  
  return x + y + z;  
}  
  
const numbers = [1, 2, 3];  
  
console.log(sum(...numbers));  
// expected output: 6
```

```
const arr1 = ['one', 'two'];  
const arr2 = [...arr1, 'three', 'four', 'five'];  
  
console.log(arr2);  
// Output:  
// ["one", "two", "three", "four", "five"]
```

```
const obj1 = { x : 1, y : 2 };  
const obj2 = { z : 3 };  
  
// add members obj1 and obj2 to obj3  
const obj3 = {...obj1, ...obj2};  
  
console.log(obj3); // {x: 1, y: 2, z: 3}
```

# Destructuring Assignment

The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

## Array Destructuring :-

```
const arrValue = ['one', 'two', 'three'];

// destructuring assignment in arrays
const [x, y, z] = arrValue;

console.log(x); // one
console.log(y); // two
console.log(z); // three
```

```
let arrValue = [10];

// assigning default value 5 and 7
let [x = 5, y = 7] = arrValue;

console.log(x); // 10
console.log(y); // 7
```

# Object Destructuring -

```
// assigning object attributes to variables
const person = {
  name: 'Sara',
  age: 25,
  gender: 'female'
}

// destructuring assignment
let { name, age, gender } = person;

console.log(name); // Sara
console.log(age); // 25
console.log(gender); // female
```

```
const person = {
  name: 'Sara',
  age: 25,
  gender: 'female'
}

// destructuring assignment
// using different variable names
let { name: name1, age: age1, gender:gender1 } = person;

console.log(name1); // Sara
console.log(age1); // 25
console.log(gender1); // female
```

# for of loop

Allows to iterate over iterable objects (arrays, strings etc).

Syntax -

```
for (element of iterable) {  
    // body of for...of  
}
```

iterable - an iterable object (array, set, strings, etc).

element - items in the iterable

```
// array  
const students = ['John', 'Sara', 'Jack'];  
  
// using for...of  
for ( let element of students ) {  
  
    // display the values  
    console.log(element);  
}
```

# class

A Class is template/blueprint for creating objects.

Ex- A sketch (prototype) of a house is a class . It contains all the details about the floors, doors, windows, etc. Based on these descriptions, you build the house. House is the object.

Syntax -

```
class myClass{  
    constructor(){  
        //  
    }  
}
```

```
class Employee{  
    constructor(id,fname, lname, salary){  
        this.id=id  
        this.fname=fname  
        this.lname=lname  
        this.salary=salary  
    }  
    set setid(x){  
        this.id=x;  
    }  
    get getid(){  
        return this.id  
    }  
    getFullName(){  
        return `${this.fname} ${this.lname}`  
    }  
    getSalary(){  
        document.write(this.salary)  
    }  
}  
const e1=new Employee(101,"Smith","Boss",20000)  
document.write(` ${e1.getFullName()} <BR>`)  
e1.setid=110  
document.write(e1.getid)
```

# Inheritance

Inheritance enables you to define a class that takes all the functionality from a parent class and allows you to add more.

## Reusability + Extensibility

```
class Manager extends Employee{
    constructor(id, fname, lname, salary, target){
        super(id, fname, lname, salary);
        this.target=target }
    getSalary(){ document.write(this.salary*1.15) }
}
const m1=new Manager(110,"dffdf","fdfnkdh",2000,1000)
document.write(` ${m1.getFullName()} <BR>`)
document.write(` ${m1.target} <BR>`)
m1.getSalary()
```

# modules



A module is nothing more than a chunk of JavaScript code written in a file. By default, variables and functions of a module are not available for use. Variables and functions within a module should be exported so that they can be accessed from within other files. Modules in ES6 work only in strict mode.

## **Exporting and importing a Module :-**

Named Exports and imports

Default Exports and imports

# Named export and import

Named exports are distinguished by their names. There can be several named exports in a module.

```
//using multiple export keyword
export component1
export component2
...
...
export componentN
```

```
//using single export keyword
export {component1,component2,...,componentN}
```

```
import {component1,component2..componentN} from module_name
```

```
import {original_component_name as new_component_name }
```

```
import * as variable_name from module_name
```

# Default export and import

Modules that need to export only a single value can use default exports.  
There can be only one default export per module.

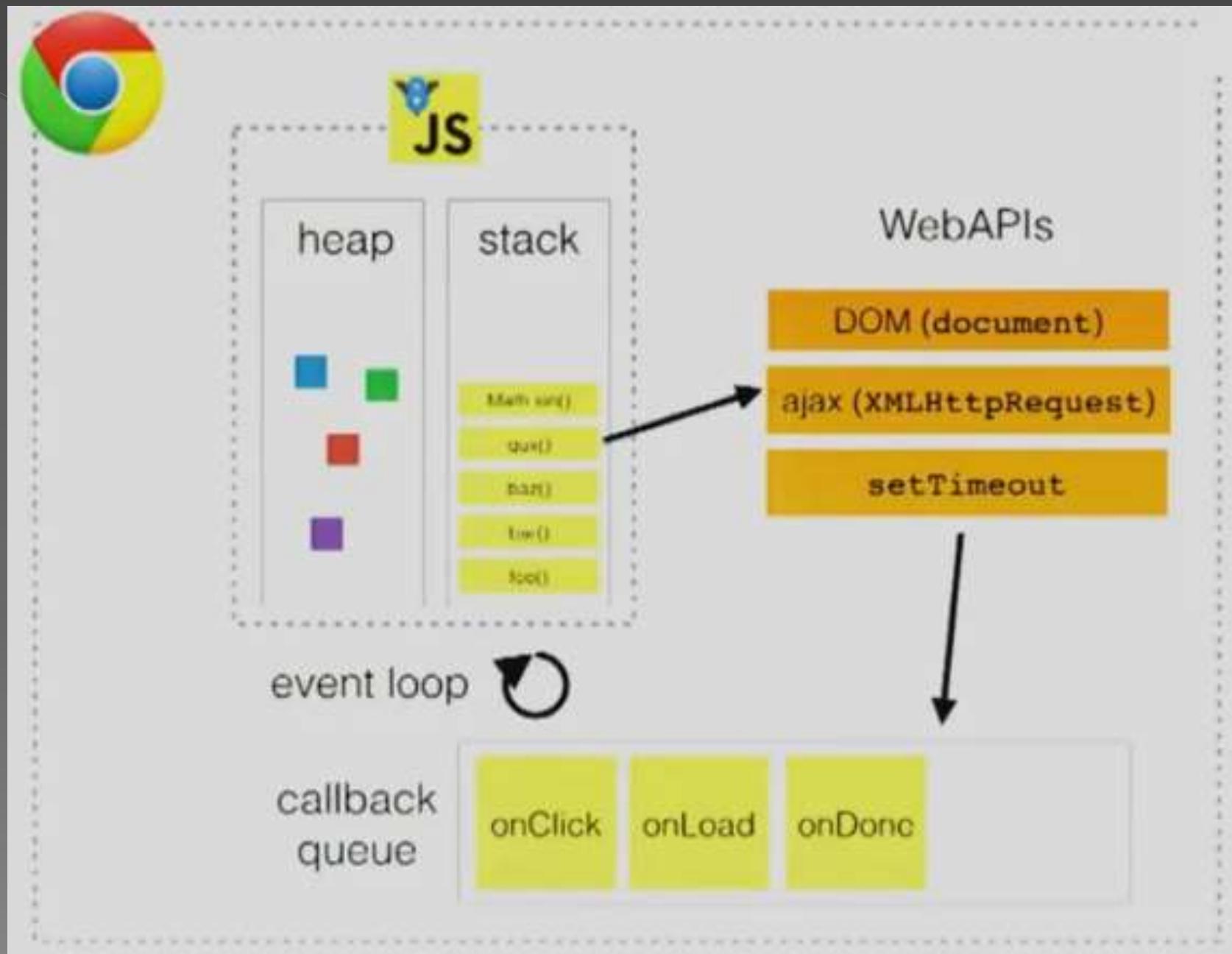
```
export default component_name
```

Unlike named exports, a default export can be imported with any name.

```
import any_variable_name from module_name
```

To execute both the modules we need to make a html file as shown below and run this in live server. Note that we should use the attribute type="module" in the script tag.

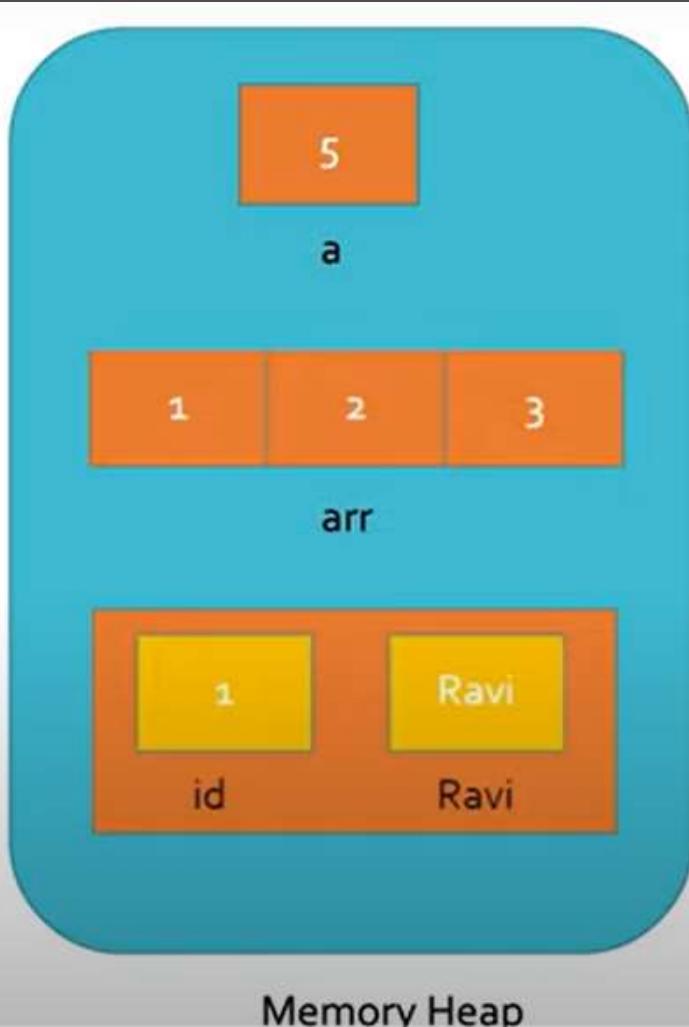
# How JavaScript Works??



# Memory Heap

- const a =5;
- const arr = [1,2,3];

```
const obj = {  
    id:1 ,  
    name: "Ravi"  
}
```



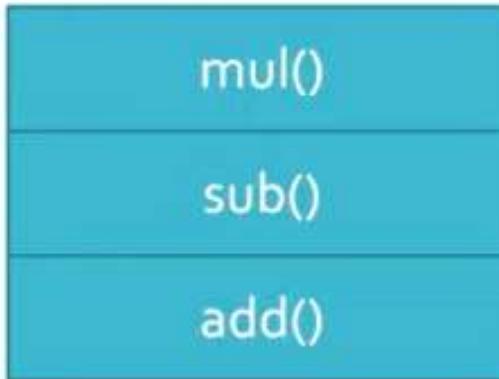
# Call stack

```
function mul(){  
    var a=1;  
}
```

```
function sub(){  
    var a=2;  
    mul();  
}
```

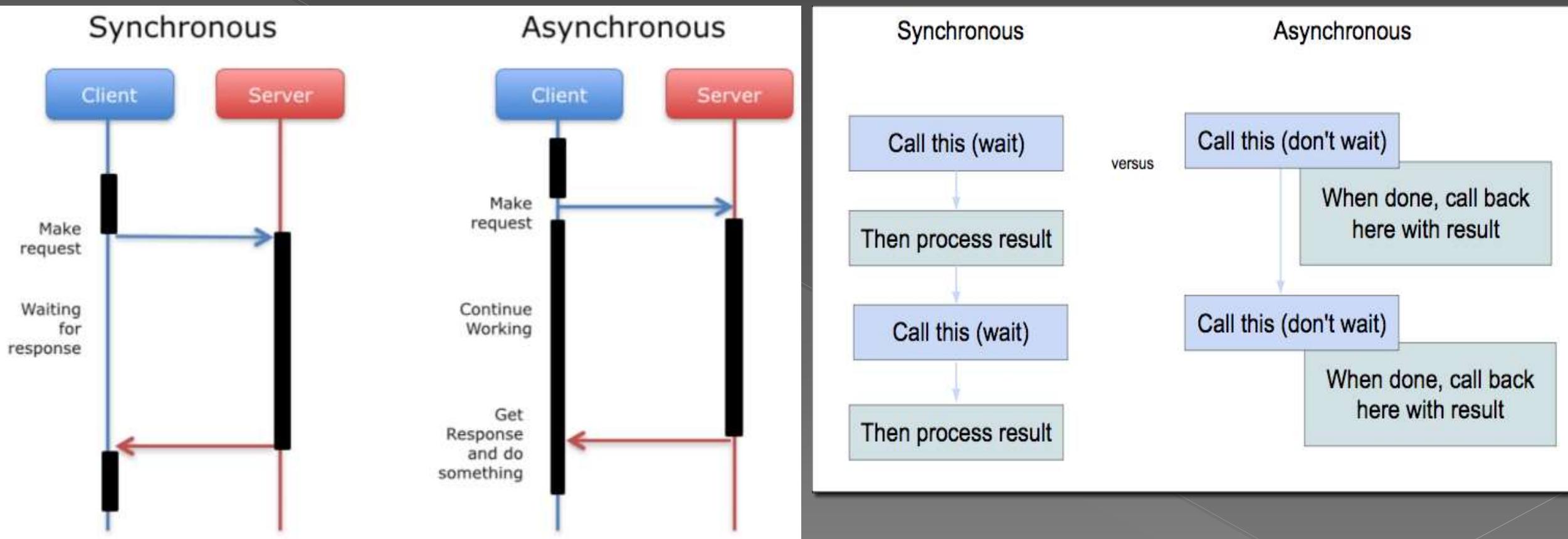
```
function add(){  
    var a=3;  
    sub();  
}
```

```
add();
```



Call Stack

# synchronous and asynchronous



# Callback

A callback is a function that is passed as an argument to another function that executes the callback based on the result. They are basically functions that are executed only after a result is produced.

Ex –

Event Handlers are sort of callbacks.  
Setitmeout and setInterval takes a callback argument

# Callback hell

Callback Hell is essentially nested callbacks stacked below one another forming a pyramid structure. Every callback depends/waits for the previous callback, thereby making a pyramid structure that affects the readability and maintainability of the code.

```
firstFunction(args, function() {  
    secondFunction(args, function() {  
        thirdFunction(args, function() {  
            // And so on...  
        });  
    });  
});
```

# Solutions to callback hell



There are four solutions to callback hell:

1. Write comments
2. Split functions into smaller functions
3. Using Promises
4. Using async/await

# Constructing a callback hell



Let's imagine we're trying to make a aloo tikki burger. To make a burger, we need to go through the following steps:

- Get ingredients
- Boil potato's
- Make aloo patties
- Get burger buns
- Put the cooked patties between the buns
- Serve the burger

# Promise

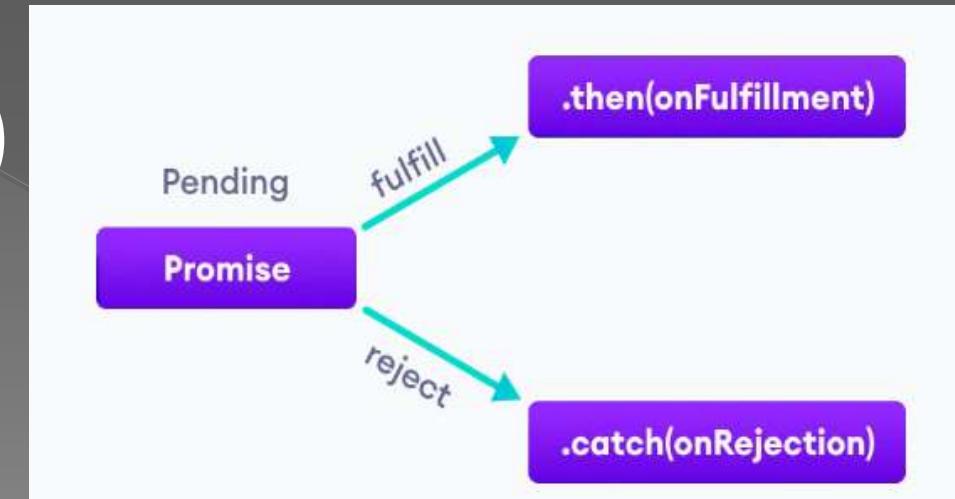
Promises can make callback hell much easier to manage.

It is an object that might return a value in the future.

It accomplishes the same basic goal as a callback function, but with many additional features and a more readable syntax.

A promise may have one of three states.

- Pending (unfulfilled yet, still being computed)
- Fulfilled (resolved , succeeded)
- Rejected (an error happened)



# Settled or Pending



A promise is settled if it is not pending (it has been resolved or rejected). Once a Promise has settled, it is settled for good. It cannot transition to any other state.

Promise users can attach callbacks to handle the fulfilled value or the reason for rejection.

# Promise with callback

Essentially, a promise is a returned object to which you attach callbacks, instead of passing callbacks into a function.

E.g., instead of an old-style function that expects two callbacks, and calls one of them on eventual completion or failure:

```
function successCallback(result) {
    console.log("It succeeded with " + result);
}

function failureCallback(error) {
    console.log("It failed with " + error);
}

doSomething(successCallback, failureCallback);
```

modern functions return a promise you can attach your callbacks to instead:

```
const promise = doSomething();
promise.then(successCallback, failureCallback)
```

# Create a Promise



```
const count = true;  
let countValue = new Promise(function (resolve, reject) {  
    if (count) {  
        resolve("There is a count value.");  
    }  
    else {  
        reject("There is no count value");  
    }  
});
```

# Consume Promise



```
countValue.then((result)=> {// executes when promise is resolved successfully  
    console.log(result);  
} )  
.catch( (error)=> { // executes if there is an error  
    console.log(error);  
} );
```

Method	Description
then()	Handles a resolve. Returns a promise, and calls onFulfilled function asynchronously
catch()	Handles a reject. Returns a promise, and calls onRejected function asynchronously
finally()	Called when a promise is settled. Returns a promise, and calls onFinally function asynchronously

# async function

An async function is a function declared with the `async` keyword, and the `await` keyword is permitted within it.

The `async` and `await` keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

Syntax -

```
async function name([param[, param[, ...param]]]) {  
    statements  
}
```

```
async function f() {  
    console.log('Async function.');//  
    return Promise.resolve(1);  
}  
  
f().then(function(result) {  
    console.log(result)  
}):
```

# await keyword

The await keyword is used inside the async function to wait for the asynchronous operation.

```
// a promise
let promise = new Promise(function (resolve, reject) {
  setTimeout(function () {
    resolve('Promise resolved');
  }, 4000);
});

// async function
async function asyncFunc() {

  // wait until the promise resolves
  let result = await promise;

  console.log(result);
  console.log('hello');

}

// calling the async function
asyncFunc();
```

# Working of async/await

```
let promise = new Promise(function (resolve, reject) { ←
  setTimeout(function () {
    resolve('Promise resolved'), 4000);
  });
}

async function asyncFunc() {
  let result = await promise; →
  console.log(result);
  console.log('hello');
}

asyncFunc();
```

calling function

waits for promise to complete

# Fetch API

The Fetch API interface allows web browser to make HTTP requests to web servers.

Syntax :

```
let promise = fetch(url, [options])
```

```
fetch('student.json').then((res)=>{
  console.log(res)
  | return res.json()
}).then(data=>{console.log(data)
}).catch(()=>console.log('error'))
```

# JSON

# What is JSON



## JavaScript Object Notation (JSON):

- Is an open standard light-weight format that is used to store and exchange data.
- Is an easier and faster alternative to XML.
- Is language independent format that uses human readable text to transmit data objects.
- Consists of objects of name/value pairs.
- Files have the extension .json.

Syntactically, JSON is similar to the code for creating JavaScript objects. Due to this similarity, standard JavaScript methods can be used to convert JSON data into JavaScript objects.

# Why use JSON



- Straightforward syntax
- Easy to create and manipulate
- Supported by all major JavaScript frameworks
- Supported by most backend technologies

# When use JSON



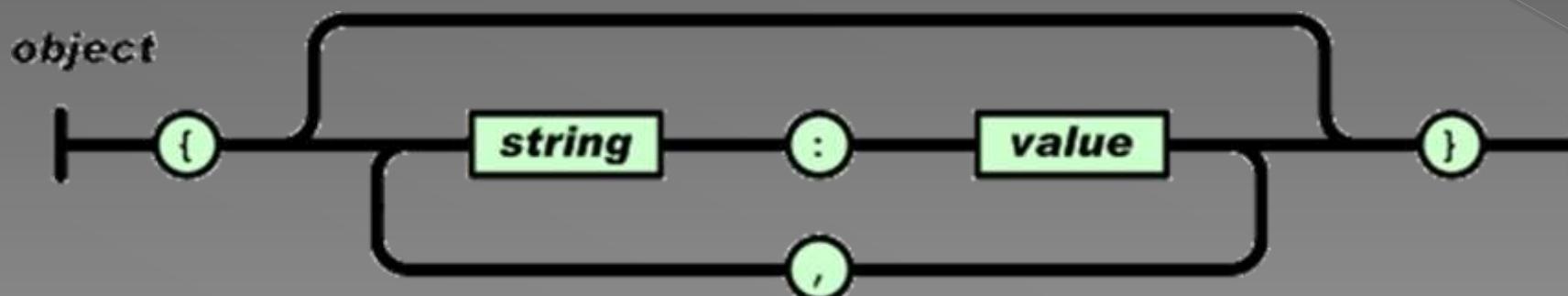
- Transfer data to and from a server
- Perform asynchronous data calls without requiring a page refresh
- Working with data stores
- Compile and save form or user data for local storage

# JSON Syntax

- JSON uses name/value pairs to store data.
- Commas are used to separate multiple data values.
- Objects are enclosed within curly braces.
- Square brackets are used to store arrays.
- JSON keys must be enclosed within double quotes.

Syntax:

```
{"fName":"Ronald", "lName":"Smith", "Contact":12112345}
```



# JSON Values

String                  Number  
Boolean                Null  
Object                Array

```
{  
  "actor": {  
    "name": "Tom Cruise",  
    "age": 56,  
    "Born At": "Syracuse, NY",  
    "Birthdate": "July 3 1962",  
    "photo": "https://jsonformatter.org/img/tom-cruise.jpg"  
  }  
}
```

```
{  
  "Actors": [  
    {  
      "name": "Tom Cruise",  
      "age": 56,  
      "Born At": "Syracuse, NY",  
      "Birthdate": "July 3, 1962",  
      "photo": "https://jsonformatter.org/img/tom-cruise.jpg"  
    },  
    {  
      "name": "Robert Downey Jr.",  
      "age": 53,  
      "Born At": "New York City, NY",  
      "Birthdate": "April 4, 1965",  
      "photo": "https://jsonformatter.org/img/Robert-Downey-Jr.jpg"  
    }  
  ]  
}
```

# JSON vs XML

## Similarities

- Both are human-readable, that is, self-describing.
- Both represent hierarchical structure, that is, values within values.
- Both can be accessed and parsed by almost every programming language.
- Both can be accessed and fetched with an XMLHttpRequest object.

## Dissimilarities

- XML needs an XML parser, whereas, a standard JavaScript method can be used to parse JSON.
- There is no need of end tag in JSON.
- JSON is much shorter as compared to XML.
- It is easy to read and write JSON.
- JSON can be used with arrays.

# JSON vs XML

## XML

```
<students>
  <student>
    < fName>Jenny</ fName>
    < lName>Watson</ lName>
  </student>
  <student>
    < fName>Dean</ fName>
    < lName>Smith</ lName>
  </student>
</students>
```

## JSON

```
{"students": [
  {"fName": "Jenny", "lName": "Watson"},
  {"fName": "Dean", "lName": "Smith"}
]}
```

# Reading Data From JSON

To read data from a JSON object, you can use the `JSON.parse()` method provided by JavaScript.

Syntax: `var obj = JSON.parse(text);`

**Example:**

```
<script>
  var x = '[      { "code": "1001", "name": "ram" },
                { "code": "1002", "name": "shyam" },
                { "code": "1003", "name": "seeta" }
  ]';
  var r = JSON.parse(x);
  for (var i in r)
    document.write(r[i].code+" "+r[i].name+"<br/>");
</script>
```

# Creating JSON Text From JavaScript

JavaScript provides you the `JSON.stringify()` method that allows you to convert JavaScript value to a JSON string.

Syntax: `var obj = JSON.stringify(value);`

```
<script>
  var x = [
    { code: "1001", name: "ram" },
    { code: "1002", name: "shyam" },
    { code: "1003", name: "seeta" }
  ];
  var r = JSON.stringify(x);
  document.write(r);
</script>
```

# Thank you !!

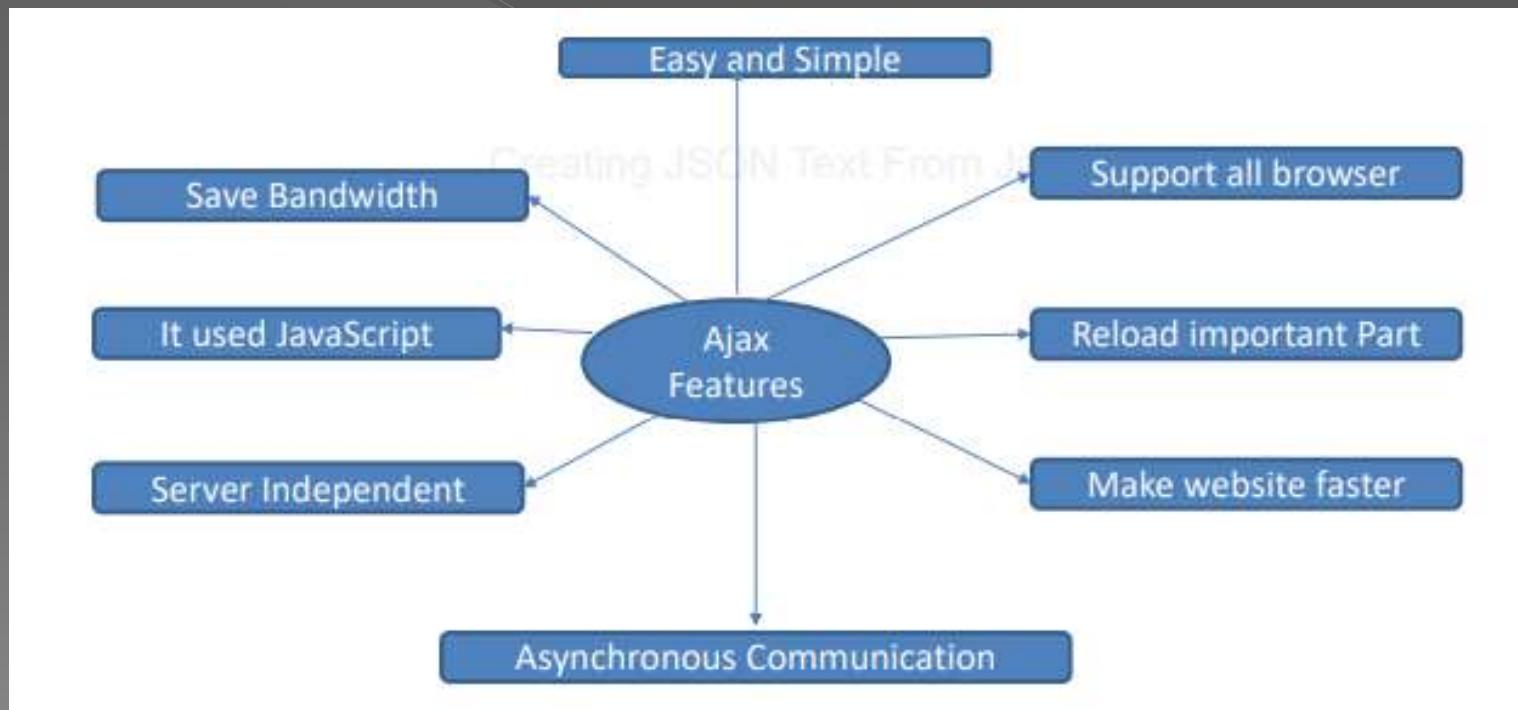
# AJAX

Harshita Maheshwari

# AJAX

Ajax stands for Asynchronous JavaScript and XML

AJAX was made popular in 2005 by Google, with Google Suggest

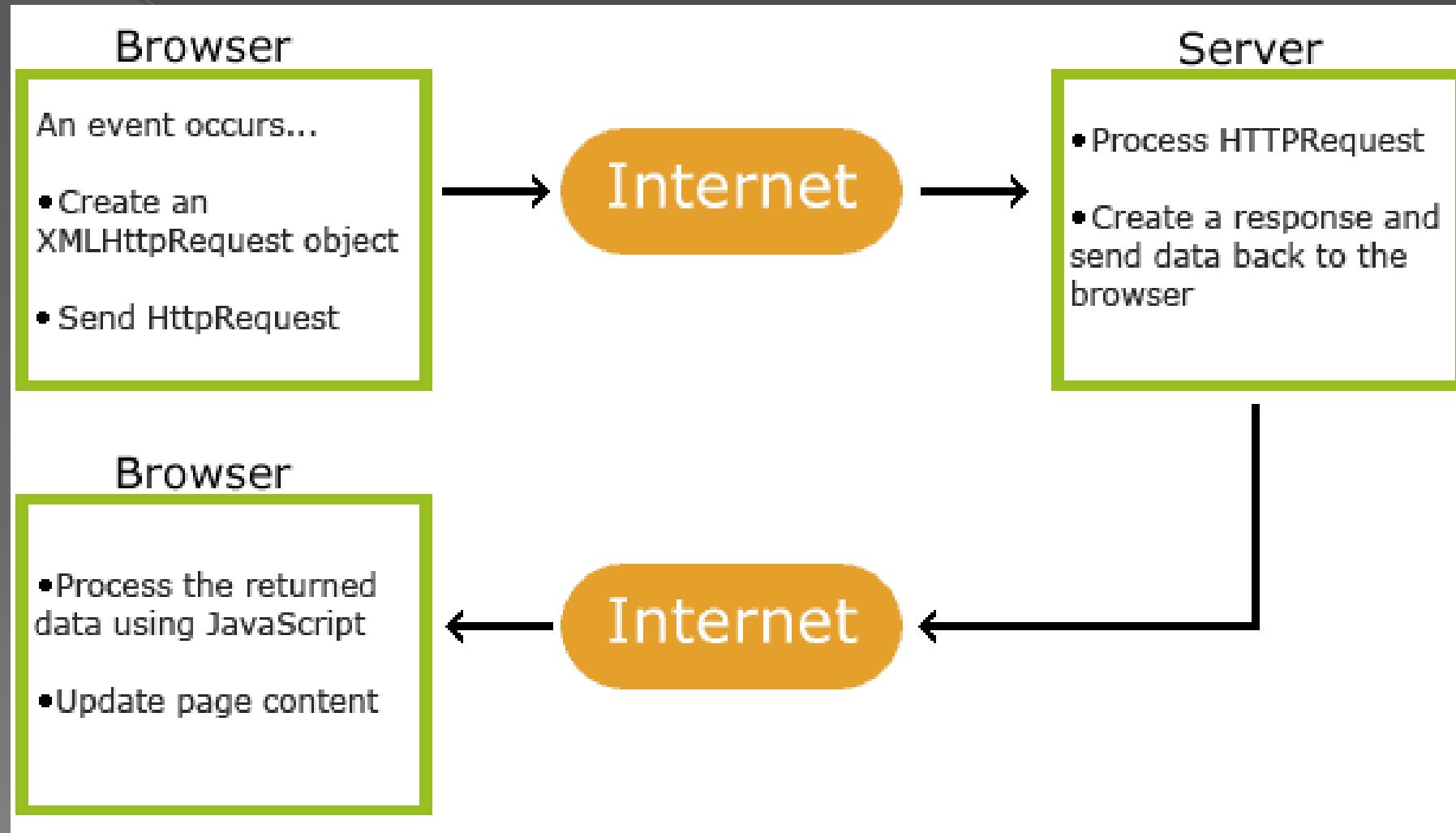


# Where should use Ajax?



- Form Validation
- Sort or Filter
- Vote or Rating
- Chat Websites
- Blog Comments
- Captcha

# AJAX Process Cycle



# Step-1

## Create a XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object. The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page without reloading the whole webpage.

Syntax:

```
Variable=new XMLHttpRequest();  
var xhttp= new XMLHttpRequest()
```

## Step-2

### Making a request to the server

The XMLHttpRequest object is used to exchange data with a server.

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object.

```
Xhttp.open("GET","ajax_info.txt",true);
```

Method used in request  
it can be get or post

This is the url of the  
file that we want to load  
into our web page

true signifies that we are making  
asynchronous request if we set it to false  
then it will not to be ajax request

# OnReadyStateChange Property



The readyState property holds the status of the XMLHttpRequest.

The onreadystatechange property defines a function to be executed

when the readyState changes.

The status property and statusText property holds the status of the XMLHttpRequest object.

readyState: Holds the status of XMLHttpRequest

0: request not initialized

1: server connection established

2: request received

3: processing request

4: request finished and response ready

Status: 200 “OK”

403 “forbidden”

404 “Page not found”

```
xhttp.onreadystatechange=function(){
If(this.readyState==4 && this.status==200){
Document.getElementById("demo").innerHTML=this.responseText;
}
};
```

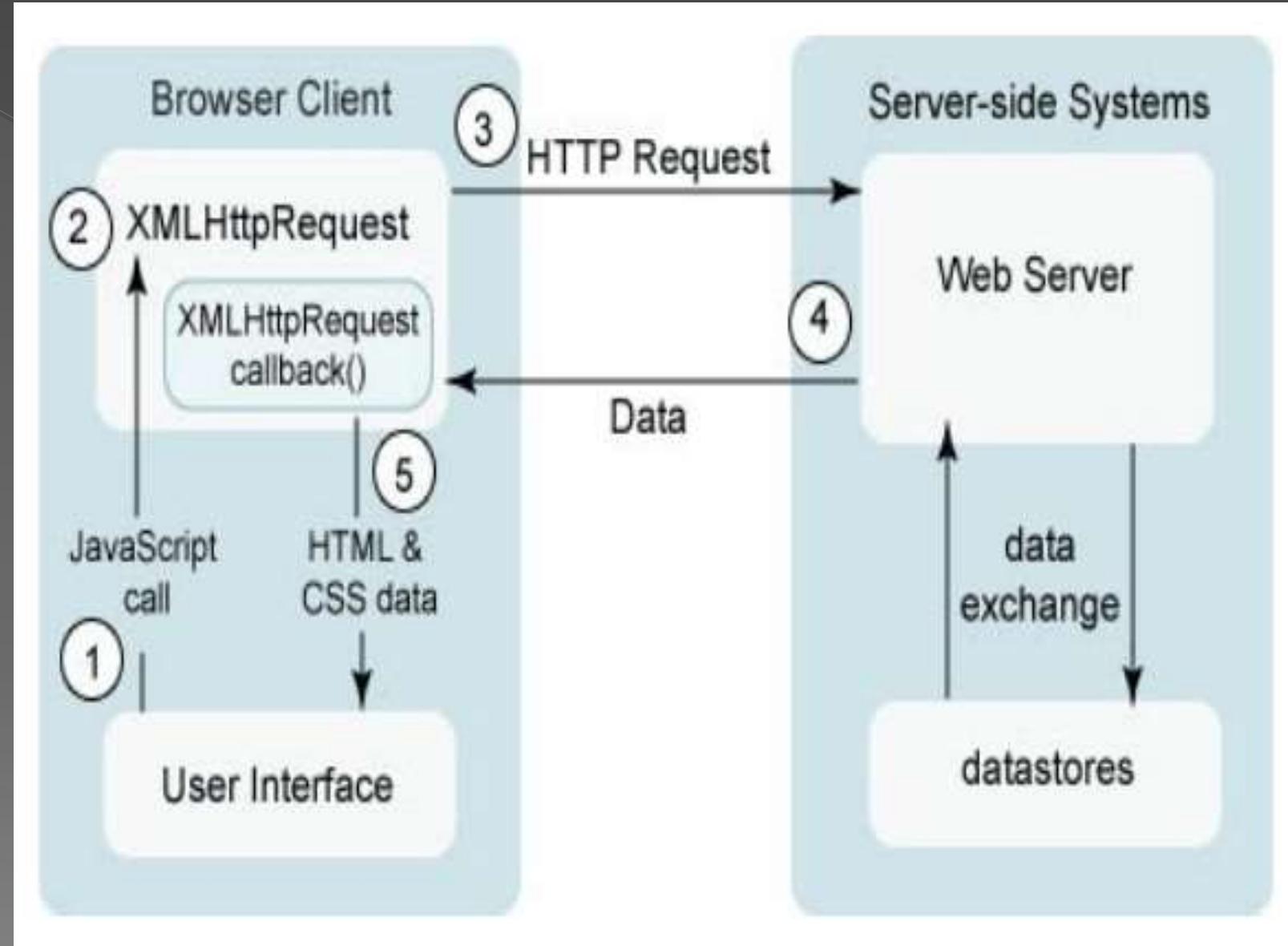
We are executing this anonymous function when this readystatechange event trigger or readystate value changes if the value of readystate is 4 and status is 200, we are just modifying the inner html of id demo with the response that is returned from the server.

# Send the request

xhttp.send()



```
var xhttp=new XMLHttpRequest();
xhttp.onreadystatechange= ()=>{
if(this.readyState==4 && this.status==200){
document.getElementById("demo").innerHTML=this.responseText;
}
};
xhttp.open("GET","ajax_info.txt",true);
xhttp.send();
```



# Handling AJAX Requests using jQuery



jQuery library provides you with various methods, known as jQuery AJAX methods, that allow you to make a call to the AJAX code. These methods allow you to perform various tasks.

The following list depicts the jQuery AJAX methods:

load()

get()

post()

ajax()

# jQuery - AJAX load() Method

The load() method is used to load or fetch data from a Web server into a selected HTML element.

Syntax :

```
$(selector).load(URL[,data][,complete])
```

URL: Is used to specify the URL from where you want to load the data. The URL can be used to refer to any resource, such as text file or html file.

data: Is used to pass an object of a key/value pair along with the request to the server.

complete: Is used to refer to a callback method that will beexecuted after the successful execution of load() method.

```
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").load("demo.txt");
    });
});
</script>
```

# jQuery \$.get() Method

The get()method is used to load data from a Web server using the HTTP GET request.

Syntax: `$.get(url[,data][, callback],[datatype])`

datatype: Is used to specify the type of data, such as text, JSON, or XML, that is expected in return from the server

```
$("button").click(function(){
    $.get("demo.html", function(data, status){
        alert("Data: " + data + "\nStatus: " + status);
    });
});
```

# jQuery \$.post() Method

The post()method is:

- Similar to the get()method.
- Used to load data from a Web server using the HTTP POST request.
- Used when the requested is large in amount.
- Used to send the data in an encrypted format.

Syntax: `$.post(url[,data][, callback],[datatype])`

```
$("button").click(function(){
    $.post("demo_post.php",
    {
        name: "Infoway Technologies",
        city: "Pune"
    },
    function(data, status){
        alert("Data: " + data + "\nStatus: " + status);
    });
});
```

# jQuery \$.ajax() Method

Syntax : `$.ajax(options)`

options: Is an optional parameter that helps in configuring the AJAX calls by using key/value pairs.

Option	Description
<code>async</code>	A boolean value that indicates whether to execute the request asynchronously.
<code>complete</code>	A callback function that executes whenever the request finishes.
<code>datatype</code>	A string defining the type of data, such as XML, HTML, JSON, or script, that is expected back from the server.
<code>success</code>	A callback function that is executed if the request succeeds.
<code>type</code>	A string defining the HTTP method to be used for the request (GET or POST).
<code>URL</code>	A required option that refers to the string containing the URL to which the request

```
$("button").click(function(){
    $.ajax({
        url: "demo.txt",
        type: 'GET',
        success: function(result){
            $("#div1").html(result);
        }
});
```

# jQuery \$.getJSON() Method

The getJSON() method is used to get JSON data using an AJAX HTTP GET request.

Syntax: `$.getJSON(url,data,success(data,status,xhr))`

`success(data,status,xhr)`:Optional. Specifies the function to run if the request succeeds  
data - contains the data returned from the server.

status - contains a string containing request status ("success", "notmodified",  
"error", "timeout", or "parsererror").

xhr - contains the XMLHttpRequest object.

```
$("button").click(function(){
    $.getJSON("demo.js", function(result){
        $.each(result, function(i, field){
            $("div").append(field + " ");
        });
    });
});
```

# jQuery \$.get() Method

The get() method is used to load data from a Web server using the HTTP GET request.

Syntax: \$(selector).get(url[,data][, callback],[datatype])

datatype: Is used to specify the type of data, such as text, JSON, or XML, that is expected in return from the server

# Thank you !!

# Node.js

Harshita Maheshwari

# Agenda for Today's Session

- **Introduction to Node.js**
- **JavaScript vs. Node.js**
- **why Node.js**
- **Event Loop**
- **Understanding Blocking I/O and Non-blocking I/O**
- **What can we do with Node?**
- **What can't do with Node?**
- **When to use Node**
- **When not use Node**
- **Node.js Environment Setup**
- **Node.js REPL**



# Introduction

**Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser. It is used for creating server-side and networking web applications.**

- **Written in C++**
- **created by Ryan Dahl starting in 2009**
- **Built on top of Chrome's V8 engine –so pure JavaScript**
- **Framework to build asynchronous I/O applications**
- **Single Threaded – no concurrency bugs –no deadlock**
- **One node process = one CPU core**
- **It is open source and free to use.**

# JavaScript vs. Node.js

JAVASCRIPT	NODEJS
Javascript is a programming language that is used for writing scripts on the website.	NodeJS is a Javascript runtime environment.
Javascript can only be run in the browsers.	NodeJS code can be run outside the browser.
It is basically used on the client-side.	It is mostly used on the server-side.
Javascript is capable enough to add HTML and play with the DOM.	Nodejs does not have capability to add HTML tags.
Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox.	Nodejs can only run in V8 engine of google chrome.
Javascript is used in frontend development.	Nodejs is used in server-side development.
Some of the javascript frameworks are RamdaJS, TypedJS, etc.	Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm.
It is the upgraded version of ECMA script that uses Chrome's V8 engine written in C++.	Nodejs is written in C, C++ and Javascript.

# Why Node.js??

- Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.
- It makes use of event-loops via JavaScript's callback functionality to implement the non-blocking I/O.

- **JavaScript used in client-side but node.js puts the JavaScript on server-side thus making communication between client and server will happen in same language**
  
- **Servers are normally thread based but Node.JS is “Event” based. Node.JS serves each request in a Evented loop that can able to handle simultaneous requests.**

# Success Stories.....



Rails to Node

- « Servers were cut to 3 from 30 »
- « Running up to 20x faster in some scenarios »
- « Frontend and backend mobile teams could be combined [...] »

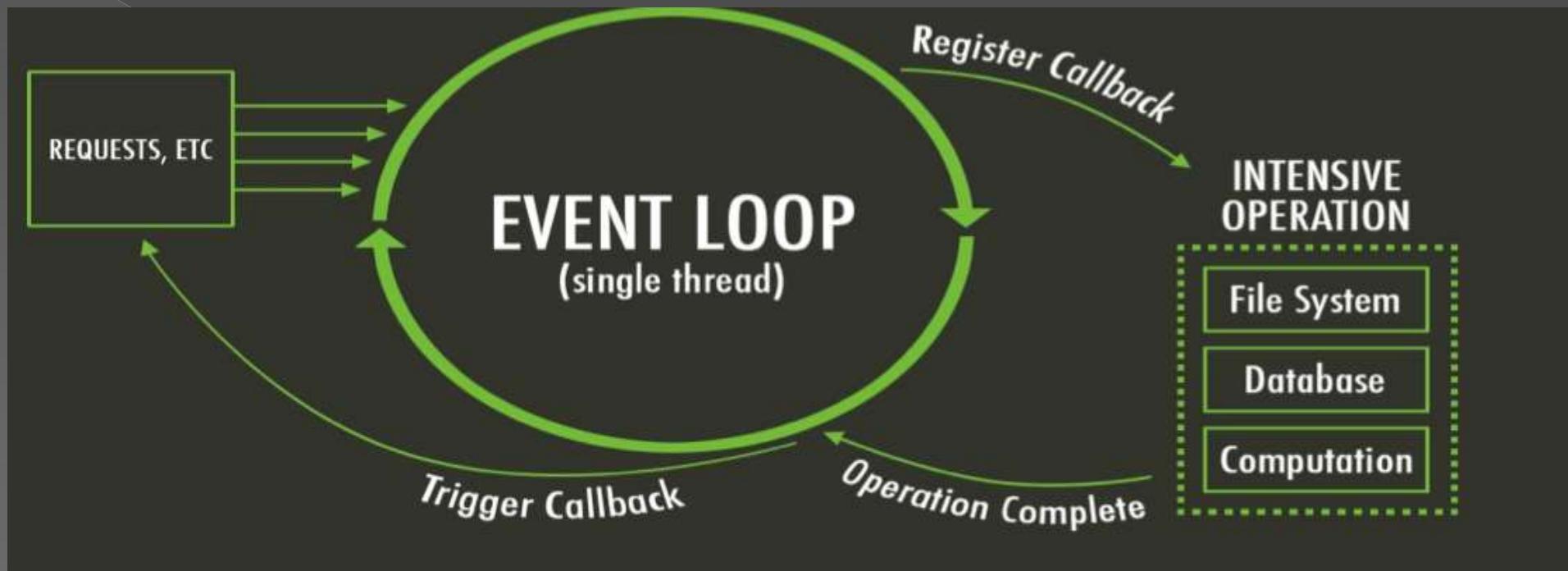


Java to Node

- « Built almost twice as fast with fewer people »
- « Double the requests per second »
- « 35% decrease in the average response time »



# Event Loop



There are a couple of implications of this apparently very simple and basic model

- Avoid synchronous code at all costs because it blocks the event loop
- Which means: callbacks, callbacks, and more callbacks

# Why node.js use event-based?



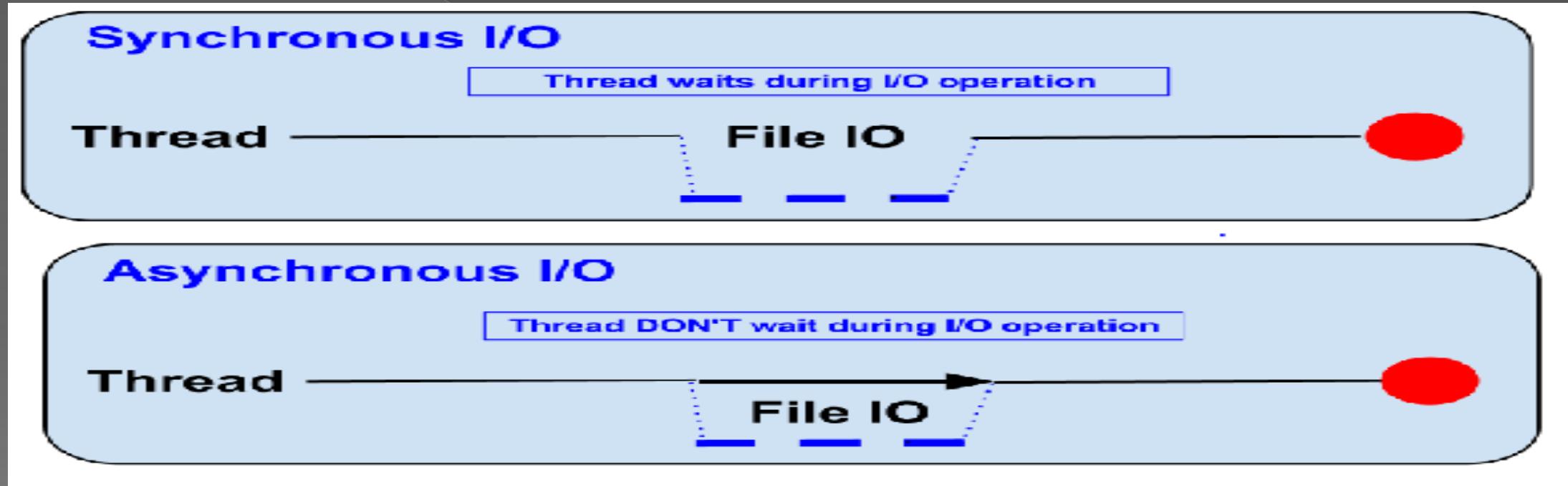
In a normal process cycle the webserver while processing the request will have to wait for the IO operations and thus blocking the next request to be processed.

Node.JS process each request as events, The server doesn't wait for the IO operation to complete while it can handle other request at the same time.

When the IO operation of first request is completed it will call-back the server to complete the request.

# Blocking vs Non-Blocking.....

Example : Read data from file and show data



# Blocking I/O

- Read data from file

- Show data

- Do other tasks

```
var data = fs.readFileSync( "test.txt" );
console.log( data );
console.log( "Do other tasks" );
```

# Non Blocking I/O

- Read data from file

When read data completed, show data



Callback

- Do other tasks

```
fs.readFile( "test.txt", function( err, data ) {  
  console.log(data);  
});
```

# What can we do with Node?



- We can create an **HTTP server** and print 'hello world' on the browser in just 4 lines of JavaScript.
- We can create a **DNS server**.
- We can create a **Static File Server**.
- We can create a **Web Chat Application** like GTalk in the browser.
- Node.js can also be used for creating **online games, collaboration tools** or anything which sends updates to the user in **real-time**.

# What can't do with Node?



- Node is a platform for writing JavaScript applications outside web browsers. This is not the JavaScript we are familiar with in web browsers. There is no DOM built into Node, nor any other browser capability.
- Node can't run on GUI, but run on terminal

# When to use Node.js

- Node.js is good for creating streaming based real-time services, web chat applications, static file servers etc.
- If you need high level concurrency and not worried about CPU-cycles.
- If you are great at writing JavaScript code because then you can use the same language at both the places: server-side and client-side.

# When not use Node.js



- Your server request is dependent on heavy CPU consuming algorithm/Job.
- Node.JS itself does not utilize all core of underlying system and it is single threaded by default, you have to write logic by your own to utilize multi core processor and make it multi threaded.

# Node.js Setup

- The following tools/SDK are required for developing a Node.js application:
  - a. Node.js
  - b. Node Package Manager (NPM)
  - c. IDE (Integrated Development Environment) or Text Editor
- **Install Node.js on Windows:**
  - a. Visit Node.js official web site <https://nodejs.org>.
  - b. It will automatically detect OS and display download.
  - c. Download node MSI for windows by clicking on Download link.
  - d. After downloading the MSI, double-click on it to start the installation.
  - e. Click Next to read and accept the License Agreement and then click Install.
  - f. It will install Node.js quickly on your computer.
  - g. Finally, click finish to complete the installation
  - h. Open command prompt and type following.  
`node -v`  
`npm -v`

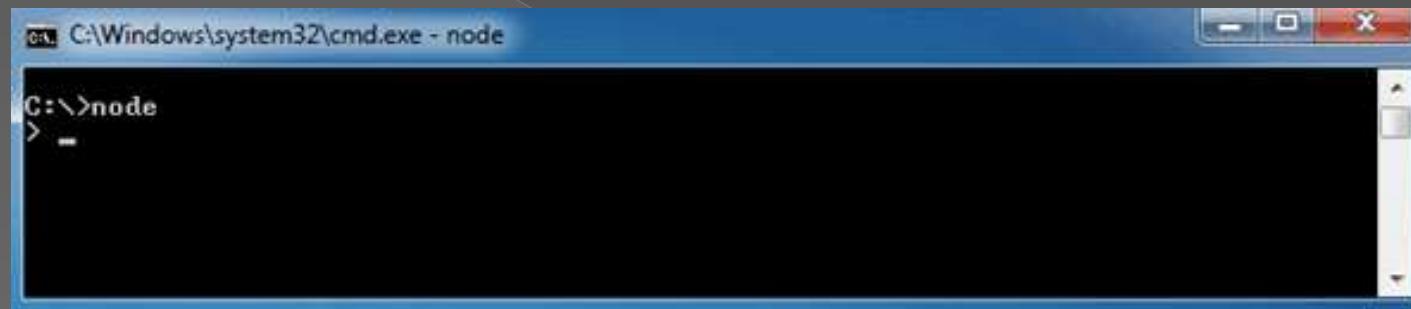
# Node.js Console - REPL



REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode. Node.js or Node comes bundled with a REPL environment. It performs the following tasks –

- Read – Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
- Eval – Takes and evaluates the data structure.
- Print – Prints the result.
- Loop – Loops the above command until the user presses ctrl-c twice.
- The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.

To launch the REPL (Node shell), open command prompt (in Windows) or terminal (in Mac or UNIX/Linux) and type node as shown below. It will change the prompt to > in Windows and MAC.



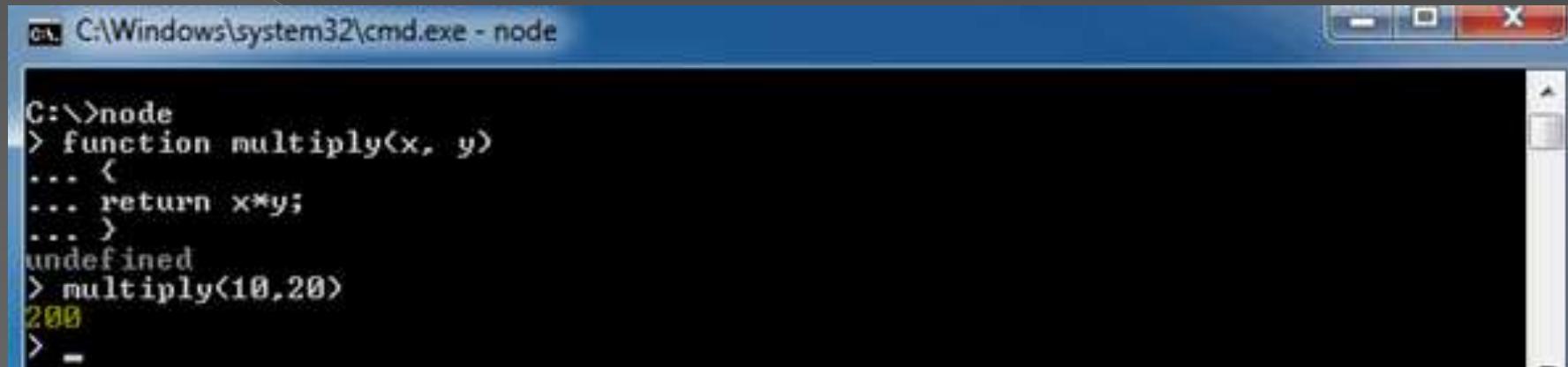
You can now test pretty much any Node.js/JavaScript expression in REPL. For example,

```
> 10 + 20  
30
```

```
> "Hello " + "World"  
Hello World
```

```
> var x = 10, y = 20;  
> x + y  
30
```

We can define a function and execute it as shown below.



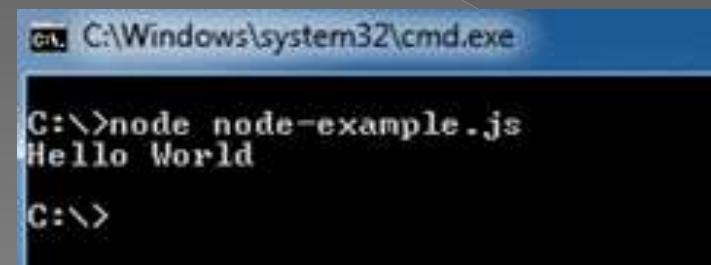
A screenshot of a Windows Command Prompt window titled "C:\Windows\system32\cmd.exe - node". The window shows the following interaction:

```
C:\>node
> function multiply(x, y)
... {
...   return x*y;
...
undefined
> multiply(10,20)
200
> _
```

You can execute an external JavaScript file by writing `node fileName` command.  
For example,

node-example.js

```
console.log("Hello World");
```



A screenshot of a Windows Command Prompt window titled "C:\Windows\system32\cmd.exe". The window shows the following command and output:

```
C:\>node node-example.js
Hello World
c:\>
```

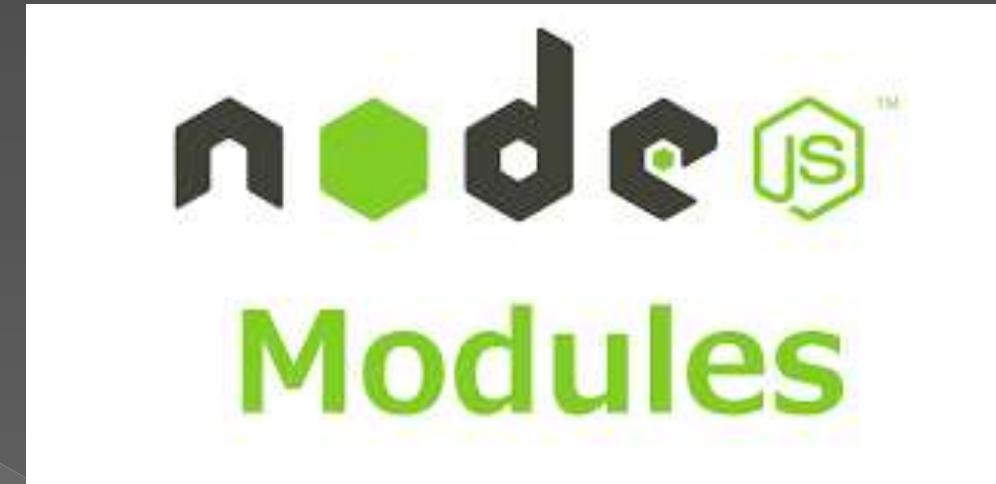
# REPL commands

The following table lists important REPL commands.

REPL Command	Description
.help	Display help on all the commands
tab Keys	Display the list of all commands.
Up/Down Keys	See previous commands applied in REPL.
.save filename	Save current Node REPL session to a file.
.load filename	Load the specified file in the current Node REPL session.
ctrl + c	Terminate the current command.
ctrl + c (twice)	Exit from the REPL.
ctrl + d	Exit from the REPL.
.break	Exit from multiline expression.
.clear	Exit from multiline expression.

# Agenda for Today's Session

- Introduction to Node.js Modules
- Types of Module
- How to create a Local Module
- How to export module
- Require function
- Introduction to NPM
- Package.json
- NPM local Package
- NPM Global Package
- Adding dependency
- Update and Search Package
- Package-lock.json



# Modules



In Node, the modularity is a first-class concept. In the Node.js module system, each file is treated as a separate module. let's say, a demo.js file is a module.

Modules help us encapsulating our code into manageable chunks. Anything that we define in our module (i.e. in our JavaScript file) remains limited to that module only, unless we want to expose it to other parts of our code.

# Types of Modules



- **Core module:** Modules that come shipped with Node.js, e.g. **https**, **os**, **fs**, **net**, etc.
- **Third-party module:** Modules that you install from any package manager. We use these modules to accomplish or simplify any existing task. For example, to simplify our web API development we use **express**, or to deal with date and time we use **moment**.
- **Local module:** These are the modules that we create for our own use. These modules basically consist of core business logic of our code.

# Global Object

- **\_\_dirname** :- It specifies the name of the directory that currently contains the code.
- **\_\_filename** : - It specifies the filename of the code being executed.
- **setImmediate(callback[, arg][, ...])**
- **setInterval(callback, delay[, arg][, ...])**
- **setTimeout(callback, delay[, arg][, ...])**
- **clearImmediate(immediateObject)**
- **clearInterval(intervalObject)**
- **clearTimeout(timeoutObject)**

# How To Create a Module



**Creating a module in Node is very simple, just create a javascript file .**

Let's say we defined one file, named sum.js, with the following content:

```
const sum = (a, b) =>
{
    return a + b;
};
const result = sum(2, 3)
console.log(result)
```

# Exports

- This is an object used to expose our functionalities in one module, so these functionalities can be used in other modules.
- We can expose anything, this can be a function, variable, constants, classes, etc.
- whatever you assign to module.exports will be exposed as a module.

Message.js

```
module.exports = 'Hello world';
```

Log.js

```
module.exports.log = function (msg) {
    console.log(msg);
};
```

data.js

```
module.exports = {
    firstName: 'James',
    lastName: 'Bond'
}
```

Log.js

```
module.exports = function (msg) {
    console.log(msg);
};
```

# Require

This is a function that we use to import or require the functionalities from other modules. It is a compliment to the exports object, which is used to export functionalities. require, on the other hand, is used to import those functionalities.

app.js

```
var msg = require('./Messages.js');

console.log(msg);
```

app.js

```
var msg = require('./Log.js');

msg.log('Hello World');
```

app.js

```
var person = require('./data.js');

console.log(person.firstName + ' ' + person.lastName);
```

app.js

```
var msg = require('./Log.js');

msg('Hello World');
```

# NPM

# Node Package Manager



- NPM is the world's largest Software Library (Registry).
- NPM is also a software Package Manager and Installer.
- The registry contains over 800,000 code packages.
- Open-source developers use npm to share software.
- Many organizations also use npm to manage private development.
- Created by Isaac Z. Schlueter

# Installing NPM



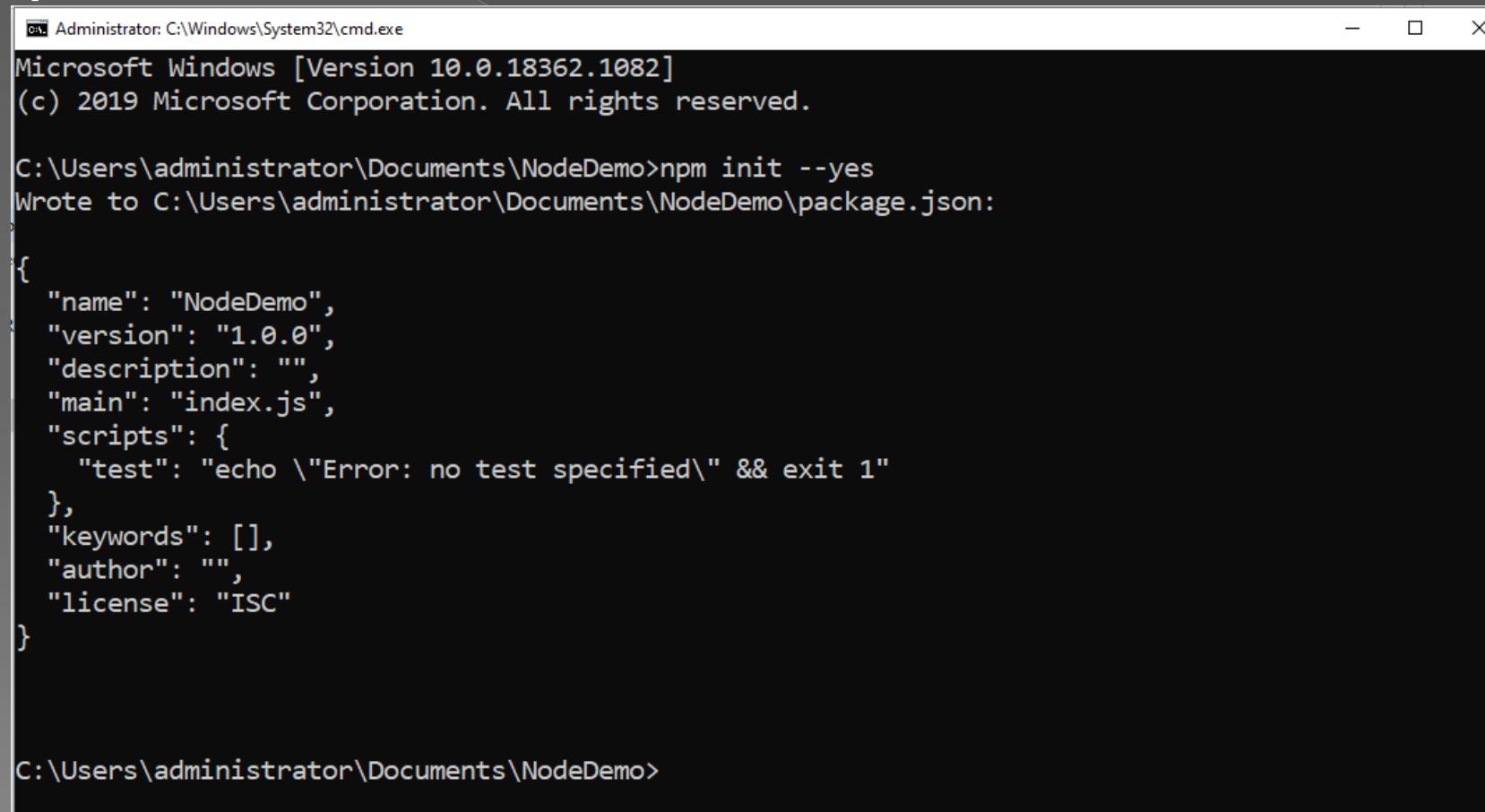
npm includes a CLI (Command Line Client) that can be used to download and install software:

```
C:\>npm install <package>
```

- npm is installed with Node.js
- All npm packages are defined in files called package.json.
- The content of package.json must be written in JSON.
- At least two fields must be present in the definition file: name and version

# Package.Json

Package.json holds various meta data relevant to the project. This file is used to give information to npm that allows it to identify project as well as handle the project's dependencies.



```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\administrator\Documents\NodeDemo>npm init --yes
Wrote to C:\Users\administrator\Documents\NodeDemo\package.json:

{
  "name": "NodeDemo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\"Error: no test specified\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

C:\Users\administrator\Documents\NodeDemo>
```

A screenshot of a Windows Command Prompt window titled "Administrator: C:\Windows\System32\cmd.exe". The window shows the output of the command "npm init --yes" run from the directory "C:\Users\administrator\Documents\NodeDemo". The output creates a "package.json" file with the following contents:  
  
{"  
 "name": "NodeDemo",  
 "version": "1.0.0",  
 "description": "",  
 "main": "index.js",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "keywords": [],  
 "author": "",  
 "license": "ISC"  
}  
  
The command prompt then returns to the directory "C:\Users\administrator\Documents\NodeDemo>".

# Attributes of Package.json

- **name** – name of the package
- **version** – version of the package
- **description** – description of the package
- **homepage** – homepage of the package
- **author** – author of the package
- **contributors** – name of the contributors to the package
- **dependencies** – list of dependencies. NPM automatically installs all the dependencies mentioned here in the node\_module folder of the package.
- **repository** – repository type and URL of the package
- **main** – entry point of the package
- **keywords** – keywords

# Installing Packages



There is a simple syntax to install any Node.js module -

**syntax:**

```
$ npm install
```

For example, following is the command to install a famous Node.js web framework module called express

```
$ npm install express
```

We have two ways to install package.

1. Install package local
2. Install package global

# Install package local



**Local packages are installed in the directory where you run `npm install`, and they are put in the `node_modules` folder under this directory.**

**By default, NPM installs any dependency in the local mode.**

**For Example: \$npm install express**

# Install package global



global packages are all put in a single place in your system  
(exactly where depends on your setup), regardless of where you run

`npm install -g <package-name>`

For example:

`$ npm install express -g`

# Adding dependency

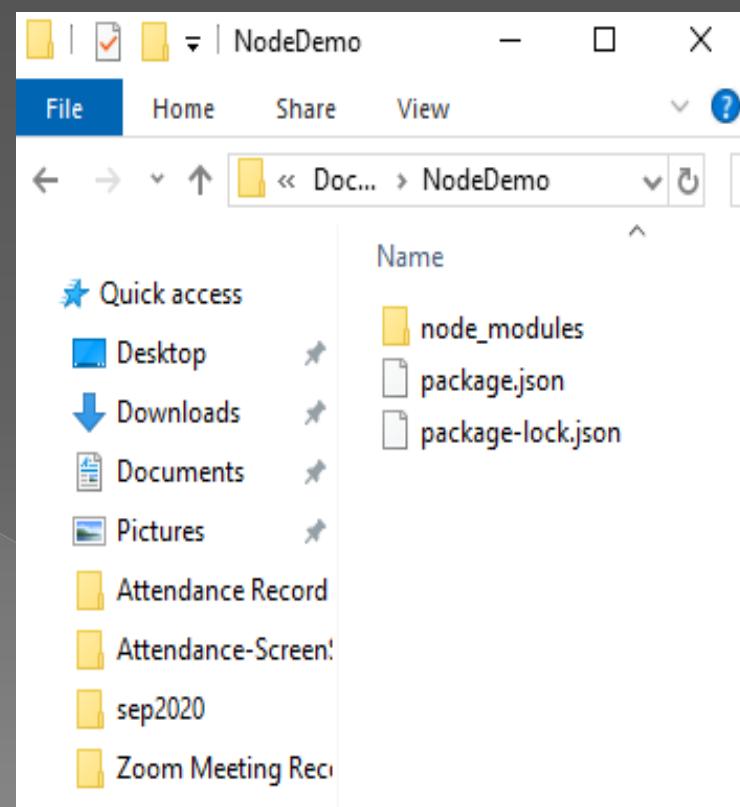
Adding dependency in package.json:

For example:

```
C:\Administrator:C:\Windows\System32\cmd.exe
C:\Users\administrator\Documents\NodeDemo>npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN NodeDemo@1.0.0 No description
npm WARN NodeDemo@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 8.363s
found 0 vulnerabilities

C:\Users\administrator\Documents\NodeDemo>
```



# Adding dependency

```
{  
  "name": "NodeDemo",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.17.1"  
  }  
}
```

# Updating packages



## ➤ **Update Package:**

Update package.json and change the version of the dependency to be updated and run the following command.

For example:

```
$ npm update express
```

## ➤ **Search a package:**

Search a package name using NPM.

For example:

```
$ npm search express
```

# package-lock.json



**package-lock.json** is automatically generated for any operations where npm modifies either the `node_modules` tree, or `package.json`. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.

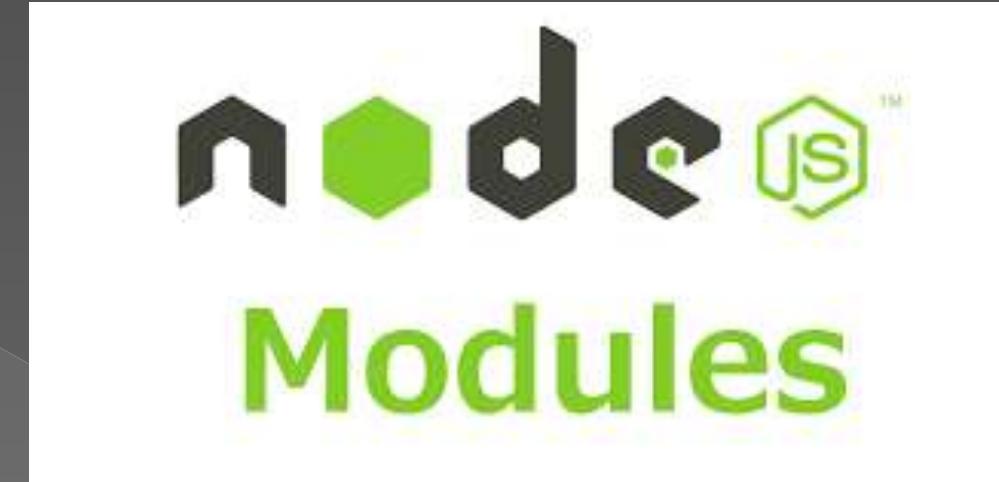
This file is intended to be committed into source repositories, and serves various purposes:

- Describe a single representation of a dependency tree such that teammates, deployments, and continuous integration are guaranteed to install exactly the same dependencies.
- Provide a facility for users to "time-travel" to previous states of `node_modules` without having to commit the directory itself.
- To facilitate greater visibility of tree changes through readable source control diffs.
- And optimize the installation process by allowing npm to skip repeated metadata resolutions for previously-installed packages.

# Agenda for Today's Session

## Core Modules-

- HTTP
- URL
- Fs
- Events



**Node.js MySQL CRUD Operations**

**Node.js MongoDB CRUD Operations**

# Core Modules



Node.js has a set of core modules that are part of the platform and come with the Node.js installation:

assert	buffer	child_process
console	cluster	crypto
dgram	dns	events
fs	http	http2
https	net	os
path	perf_hooks	querystring
readline	repl	stream
string_decoder	timers	tls
tty	url	util
v8	vm	wasi
worker	zlib	

# HTTP Module



Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

Syntax:

```
const http = require('http');
```

//create a server object:

```
http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write('Hello'); //write a response to the client  
    res.end(); //end the response  
}).listen(1000); //the server object listens on port 1000
```

# URL Module



The URL module splits up a web address into readable parts.

```
var url = require('url');
var adr = 'http://localhost:1000/default.htm?id=2&name=ram';
var q = url.parse(adr, true);
```

```
console.log(q.host); //returns 'localhost:1000'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?id=2&name=ram'
```

```
var qdata = q.query; //returns an object: { id: 2, name: 'ram' }
console.log(qdata.name); //returns 'ram'
```

# fs Module



The File System module provides a way of working with the computer's file system.

Syntax:

```
const fs = require('fs');
```

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

# Reading File

## Reading File Asynchronously:-

```
var fs = require('fs');

  fs.readFile('TestFile.txt', function (err, data)
  {
    if (err) throw err;
    console.log(data);
 });
```

## Reading File Synchronously:

```
var fs = require('fs');

var data = fs.readFileSync('dummyfile.txt', 'utf8');
console.log(data);
```

# Writing and Append Data File



## Creating & Writing File:-

```
var fs = require('fs');
fs.writeFile('test.txt', 'Hello World!', function (err)
{ if (err) console.log(err);
else
console.log('Write operation complete.');
});
```

## Append File Content:-

```
var fs = require('fs');
fs.appendFile('test.txt', 'Hello World!', function (err) {
if (err) console.log(err);
else
console.log('Append operation complete.');
});
```

# Delete File



```
var fs = require('fs');
fs.unlink('test.txt', function () {
  console.log('write operation complete.');
});
```

# Events



**Node.js allows us to create and handle custom events easily by using events module. Event module includes EventEmitter class which can be used to raise and handle custom events.**

```
// get the reference of EventEmitter class of events module  
var events = require('events');
```

```
//create an object of EventEmitter class by using above reference  
var em = new events.EventEmitter();
```

```
//Subscribe for FirstEvent  
em.on('FirstEvent', function (data) {  
  console.log('First subscriber: ' + data);});
```

```
// Raising FirstEvent  
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

# END

# Thank you !!

# Express.js

Harshita Maheshwari

# Introduction

Fast, unopinionated, minimalist web  
framework for Node.js

# What is Express??



- Express.js is a very popular web application framework built to create Node.js Web based applications.
- It provides an integrated environment to facilitate rapid development of Node based Web applications.
- Express framework is based on Connect middleware engine and used Jade html template framework for HTML templating.
- Developed by TJ Holowaychuk in Nov 2010

# Why use Express??

- Express lets you build single page, multi-page, and hybrid web and mobile applications. Other common backend use is to provide an API for a client (whether web or mobile).
- It comes with a default template engine, Jade which helps to facilitate the flow of data into a website structure and does support other template engines.
- It supports MVC (Model-View-Controller), a very common architecture to design web applications.
- It is cross-platform and is not limited to any particular operating system.
- It leverages upon Node.js single threaded and asynchronous model.

# Advantages

- Makes Node.js web application development fast and easy.
- Easy to configure and customize.
- Allows you to define routes of your application based on HTTP methods and URLs.
- Includes various middleware modules which you can use to perform additional tasks on request and response.
- Easy to integrate with different template engines like Jade, Vash, EJS etc.
- Allows you to define an error handling middleware.
- Easy to serve static files and resources of your application.
- Allows you to create REST API server.
- Easy to connect with databases such as MongoDB, Redis, MySQL

# Install Express



To install Express.js, first, you need to create a project directory and create a package.json file which will be holding the project dependencies. Below is the code to perform the same:

**npm init**

Now, you can install the express.js package in your system. To install it globally, you can use the below command:

**npm install -g express**

Or, if you want to install it locally into your project folder, you need to execute the below command:

**npm install express --save**

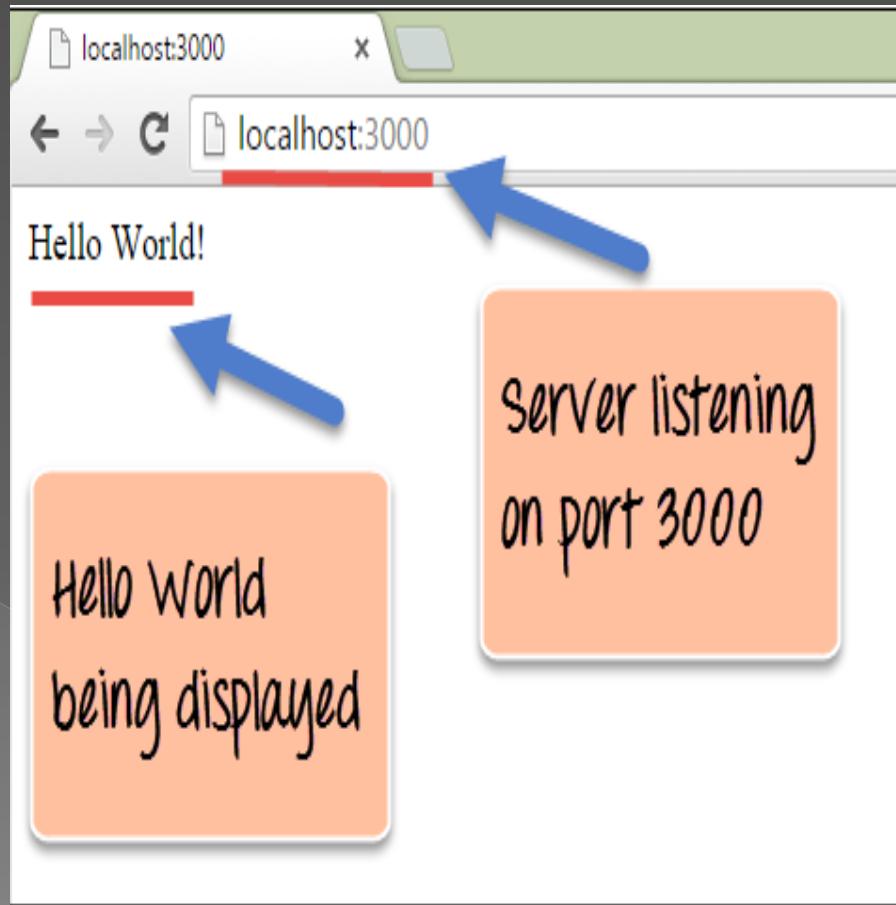
# Simple Server on Express.js

```
var express=require('express'); ①
var app=express();
app.get('/', function (req, res) { ②
  res.send('Hello World!');
})
var server = app.listen(3000, function () { ③
  console.log('Hello World!'); ④
}) ⑤
```

use the express module

Create a callback function

Send 'Hello World' response



# Express.js Request Object

The express.js request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on

Properties	Description
req.body	It contains key-value pairs of data submitted in the request body. By default, it is undefined, and is populated when you use body-parsing middleware such as body-parser.
req.params	An object containing properties mapped to the named route ?parameters?. For example, if you have the route /user/:name, then the "name" property is available as req.params.name. This object defaults to {}.
req.path	It contains the path part of the request URL
req.query	An object containing a property for each query string parameter in the route.
req.route	The currently-matched route, a string.
req.cookies	When we use cookie-parser middleware, this property is an object that contains cookies sent by the request.

# Express.js Response Object

The Response object (res) specifies the HTTP response which is sent by an Express app when it gets an HTTP request.

## Response Object Methods:-

method	Description
res.end()	End the response process
res.json()	Send a JSON response
res.redirect()	Redirect a request
res.render()	Render a review template
res.send()	Send a response of various types
res.sendFile()	Send a file as an octet stream
res.sendStatus()	Set the response status code and send its string representation as the response body

# Routing

Routing determine the way in which an application responds to a client request to a particular endpoint. which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

Each route can have one or more handler functions, which are executed when the route is matched.

The general syntax for a route is shown below: -

**app.METHOD(PATH, HANDLER)**

Wherein,

- 1) app is an instance of the express module
- 2) METHOD is an HTTP request method (GET, POST, PUT or DELETE)
- 3) PATH is a path on the server.
- 4) HANDLER is the function executed when the route is matched.

# HTTP Methods

Method	Description
1. GET	The HTTP GET method helps in requesting for the representation of a specific resource by the client. The requests having GET just retrieves data and without causing any effect.
2. POST	The HTTP POST method helps in requesting the server to accept the data that is enclosed within the request as a new object of the resource as identified by the URI.
3. PUT	The HTTP PUT method helps in requesting the server to accept the data that is enclosed within the request as an alteration to the existing object which is identified by the provided URI.
4. DELETE	The HTTP DELETE method helps in requesting the server to delete a specific resource from the destination.

# Route paths based on strings



**This route path will match requests to the root route, /.**

```
app.get('/', function (req, res) {  
  res.send('root path');});
```

**This route path will match requests to /about.**

```
app.get('/about', function (req, res) {  
  res.send('about us page');});
```

**This route path will match requests to /random.text.**

```
app.get('/random.text', function (req, res) {  
  res.send('random.text file content');});
```

# **express.Router**



- Use the **express.Router** class to create modular, mountable route handlers.
- Over the period of time routes grow in size and is extremely difficult to manage.
- Using modular approach using Router we can easily develop, maintain and extend routes.
- We will need to get the Router object and then create routes for the modules.

## Create one folder -> routes -> products.js

```
var express = require("express");

var router = express.Router();

// /products/
router.get('/', (req, res)=> {
    res.send("Get Request for Products");
});

// /products/get-product-details
router.get('/get-product-details', (req, res)=> {
    res.send("Get Request for Specific Product");
});

module.exports = router;
```

## outside the routes folder -> index.js

```
var express = require("express");

var products = require('./routes/products');
var app = express();

app.use('/products', products);
```

# Dynamic Routes

Express allows us to build URL's dynamically.

```
router.get('/user-details/:id', (req, res)=> {
  res.send("Get Request for Specific User"+req.params.id);
});
```

```
router.get('/search-by-location/:state/:city', (req, res)=> {
  res.send("Get Request for Specific User"+req.params.state + req.params.city);
});
```

# URL Binding using regex-

```
router.get('/search/:key([0-9]{4})', (req, res)=>{
  res.send("Data captured is "+req.params.key);
});
```

```
router.get('/search-username/:key([a-zA-Z]{4})', (req, res)=>{
  res.send("Data captured is "+req.params.key);
});
```

## Wild Card Route -

```
router.get('*', (req, res)=> [
  res.send("URL not found");
])
```

# Middleware

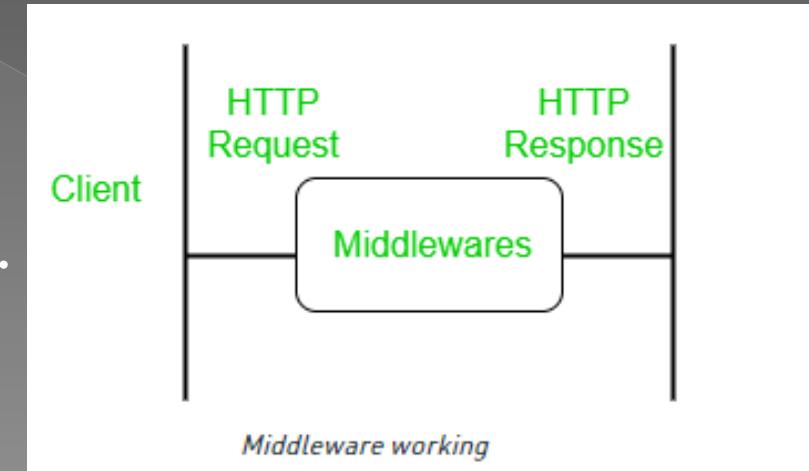


- Middleware are different types of functions that are invoked by the Express.js routing layer before the final request handler. As the name specified, Middleware appears in the middle between an initial request and final intended route. In stack, middleware functions are always invoked in the order in which they are added.
  
- Middleware is commonly used to perform tasks like body parsing for URL-encoded or JSON requests, cookie parsing for basic cookie handling, or even building JavaScript modules on the fly.

# Middleware Function

- middleware functions are the functions which have access to the request and response objects along with the next function present in the application's request-response cycle.
- Middleware can process request objects multiple times before the server works for that request.
- Middleware can be used to add logging and authentication functionality.
- Middleware improves client-side rendering performance.
- Middleware is used for setting some specific HTTP headers.
- Middleware helps for Optimization and better performance.

**Order of methods is extremely important.**



```
var express = require("express");
var router = express.Router();
router.use('/', (req, res, next)=> {
    console.log("API call received");
    next();
});
router.get('/', (req, res)=> {
    res.send("Get Request for Users");
});
router.get('/', (req, res, next)=> {
    res.send("Get Request for Users");
    next();
});
router.use('/', (req, res)=> {
    console.log("API call ended");
});
router.use('/', (req, res, next)=> {
    req.headers["content-type"]='application/json';
    console.log("API call received");
    next();
});
router.get('/', (req, res, next)=> [
    res.send("Headers Received" + req.headers["content-type"]),
    res.send("Get Request for Users" ),
    next(),
]);
const middleware = (req,res, next) => {
    console.log(`Hello my Middleware`);
    next();
}
app.get('/', (req, res) => {
    res.send(`Hello world from the server`);
});
app.get('/about', middleware, (req, res) => {
    res.send(`Hello About world from the server`);
});
```

# Express Generator



**Application generator tool to quickly create an application skeleton.**

**Easily get standard application shell for quick and rapid prototyping.**

**How to use ->**

**Install Globally -> npm i -g express-generator**

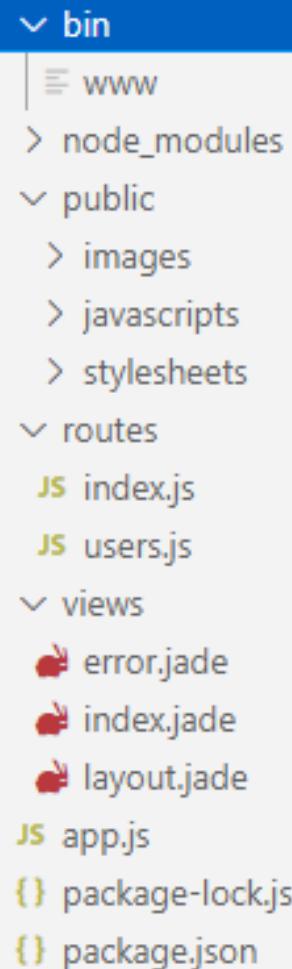
**Install locally -> npm i express-generator**

# How to create project using express-generator



- **express projectname**
- **Go to project folder and install dependencies-> npm install or npm i**
- **Start the node server -> npm start**
- **Go to browser and write localhost:3000 in the address bar to execute the express code**

# Project Structure



- app.js:- This file starts your web server. All your set up logic should be in this file.
- Public:- All the public files such as images, javascript files, CSS files should go into this folder.
- Routes:- All your routing-related logic should go into this folder.
- Views:- So this folder contains all your views i.e. HTML/hbs files. Drop this folder if you are building rest API's.

# Template Engine



- A template engine enables you to use static template files in your application.
- At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client.
- This approach makes it easier to design an HTML page.
- By default - jade

# Types of engines

Pug (formerly known as jade)	handlebars	haml-coffee
Mustache	hogan	ect
Dust	jazz	ejs
Atpl	jtpl	haml
Eco	hbs	JUST

JS index.js ×

routes > JS index.js > ...

```
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET home page. */
5 router.get('/', function(req, res, next) {
6   res.render('index', { title: 'Express' });
7 });
8
9 module.exports = router;
```

▷ □ ...

index.jade ×

views > index.jade

```
1 extends layout
2
3 block content
4 h1= title
5 p Welcome to #{title}
```

```
/* http://localhost:3000/getname?name=harshita */
router.get('/getname', function(req, res, next) {
  res.render('index', { name: req.query.name });
});
```

```
/* http://localhost:3000/test/10 */
router.get('/test/:id', function(req, res, next) {
  console.log(req.params)
  res.render('index', { id: req.params.id });
});
```

h1=name

h1=id

# Express handlerbars Templating Engine

- Uninstall jade -> npm uninstall jade –save
- Install hbs -> npm install hbs –save
- In app.js => change view engine from jade to hbs
  - `app.set('view engine', 'hbs');`
- Change extension of all view files from jade to hbs and change the files content as well.

```
{{!-- layout.hbs --}}
<!doctype html>
<html>
  <head><title>Express App</title>
  <link rel="stylesheet" href="stylesheets/style.css">
  </head>
  <body>
    {{{body}}}
  </body>
</html>
```

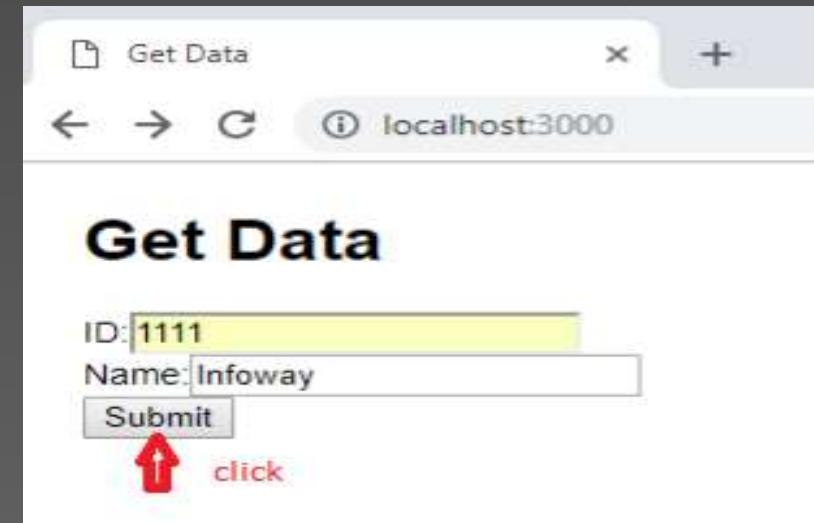
```
{{!-- index.hbs --}}
<h1>{{title}}</h1>
<p>Welcome to {{title}}</p>
```

# Handling Form Data

```
<h1>{{title}}</h1>
<form method="POST" action="/test/submit">
ID:<input type="text" name="id"><br>
Name:<input type="text" name="name"><br>
<button>Submit</button>
</form>
<br>
<br>
{{id}} {{name}}
```

```
var express = require('express');
var router = express.Router();

router.get("/",function(req,res,next){
  res.render('index', {title:'Get Data'});
});
router.post('/test/submit', function(req, res, next) {
  res.send(req.body.id+" "+req.body.name);
//res.render('index', {title:'data', id:req.body.id,name:req.body.name });
});
module.exports = router;
```



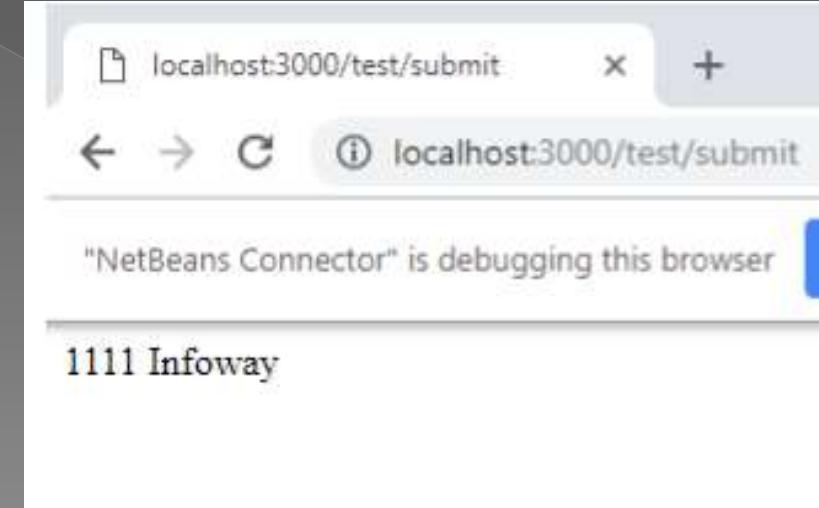
Get Data

ID: 1111

Name: Infoway

Submit

click



localhost:3000/test/submit

"NetBeans Connector" is debugging this browser

1111 Infoway

# Main inconveniences of native MongoDB Driver



- No data validation
- No casting during inserts
- No encapsulation
- No references (joins)

# Mongoose- ODM for Node.js



- Although MongoDB won't impose an structure, applications usually manage data with one. We receive data and need to validate it to ensure what we received is what we need. We may also need to process the data in some way before saving it. This is where Mongoose kicks in.
- Mongoose is an NPM package for NodeJS applications. It allows to define schemas for our data to fit into, while also abstracting the access to MongoDB. This way we can ensure all saved documents share a structure and contain required properties.

# Connect mongoose with express server



Install mongoose -> npm install mongoose --save

Note:- mongoDB server should be started.

# Mongoose setup

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mycollection');

var Schema = mongoose.Schema;

var PostSchema = new Schema({
  title: { type: String, required: true },
  body: { type: String, required: true },
  author: { type: ObjectId, required: true, ref: 'User' },
  tags: [String],
  date: { type: Date, default: Date.now }
});

mongoose.model('Post', PostSchema);
```

Validation of presence

Reference

Simplified declaration

Default value

# Schema types

- **String**
- **Number**
- **Date**
- **Buffer**
- **Boolean**
- **Mixed**
- **ObjectId**
- **Array**
- **Map**
- **Schema**

**required**: boolean or function, if true adds a required validator for this property

**default**: Any or function, sets a default value for the path. If the value is a function, the return value of the function is used as the default.

**select**: boolean, specifies default projections for queries

**validate**: function, adds a validator function for this property

**get**: function, defines a custom getter for this property

**set**: function, defines a custom setter for this property.

# Third Party Middleware

Middleware Module	Description
body-parser	Parse HTTP Request
cookie-parser	Parse cookie header and populate request cookies.
cors	Enable cross-origin resource sharing (CORS) with various options.
errorhandler	Development error-handling /debugging
morgan	HTTP Request logger
multer	Handle multi-part form data
serve-static	Serve static files
session	Establish server based sessions
timeout	Set a timeout period for HTTP request processing

# END

# Thank you !!

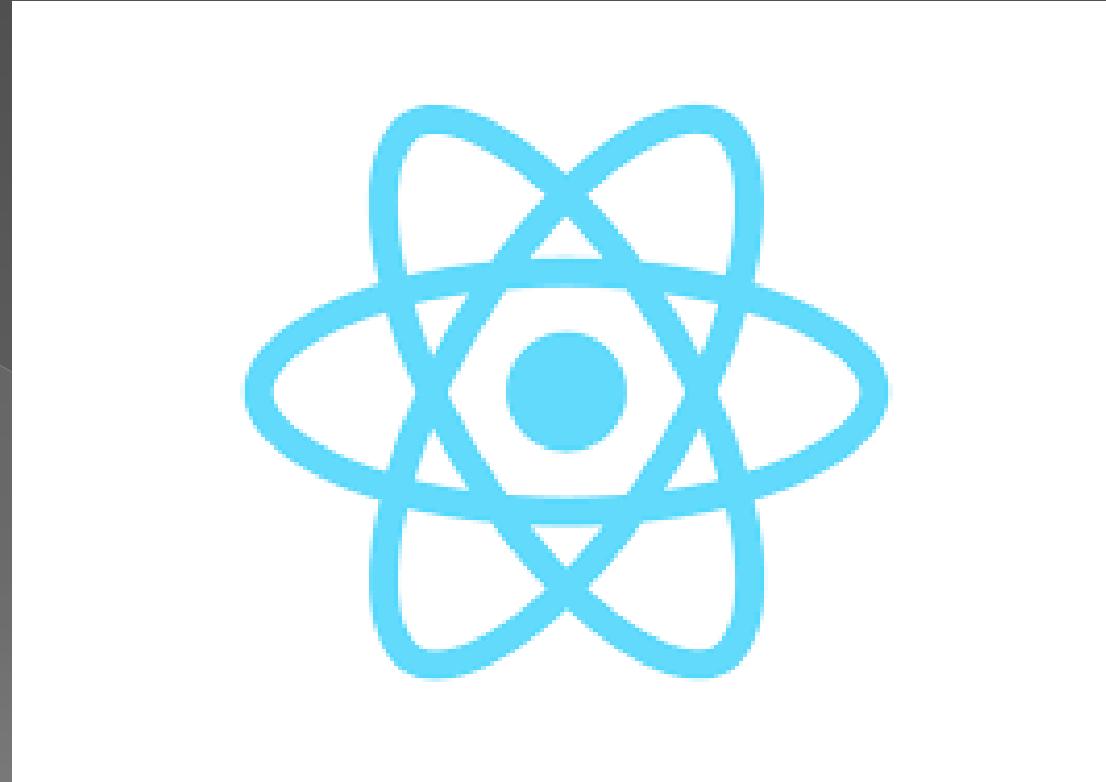
# Web Programming Technologies

## ReactJS

Harshita Maheshwari

# Agenda for Today's Session

- Introduction to ReactJS
- Why ReactJS
- How to install
- Features
- Component – Functional and Class
- Props
- State
- Component LifeCycle Methods
- Event Handling
- Styling with CSS Basic
- Conditional Rendering
- List Rendering
- Fragment
- Form Handling
- Refs
- Error Boundary
- Context



# Introduction

**React is a JavaScript library for building fast and interactive user interface.**

- Open-source
- Reusable component-based front-end library
- In a model view controller architecture, React is the “View” which is responsible for how the app looks and feels.
- Rich ecosystem
- Not a framework
- Focus on UI
- Created by Jordan Walke, who was a software engineer at Facebook

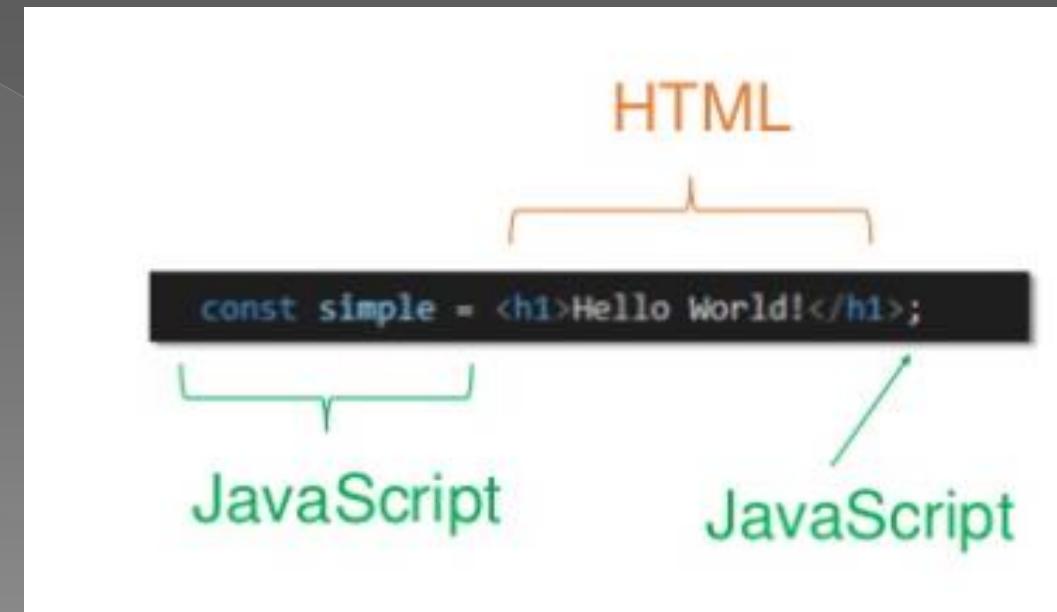
# Features



- JSX (JavaScript Syntax Extension)
- Virtual DOM
- Performance
- Extensions
- Debugging
- Component

# JSX

- JSX is a syntax extension to JavaScript. It is used with React to describe what the UI should look like.
- By using JSX, we can write HTML structure in the same file that contains JavaScript Code.
- JSX is not supported by the browsers, as a result Babel compiler transpile the code into JavaScript code

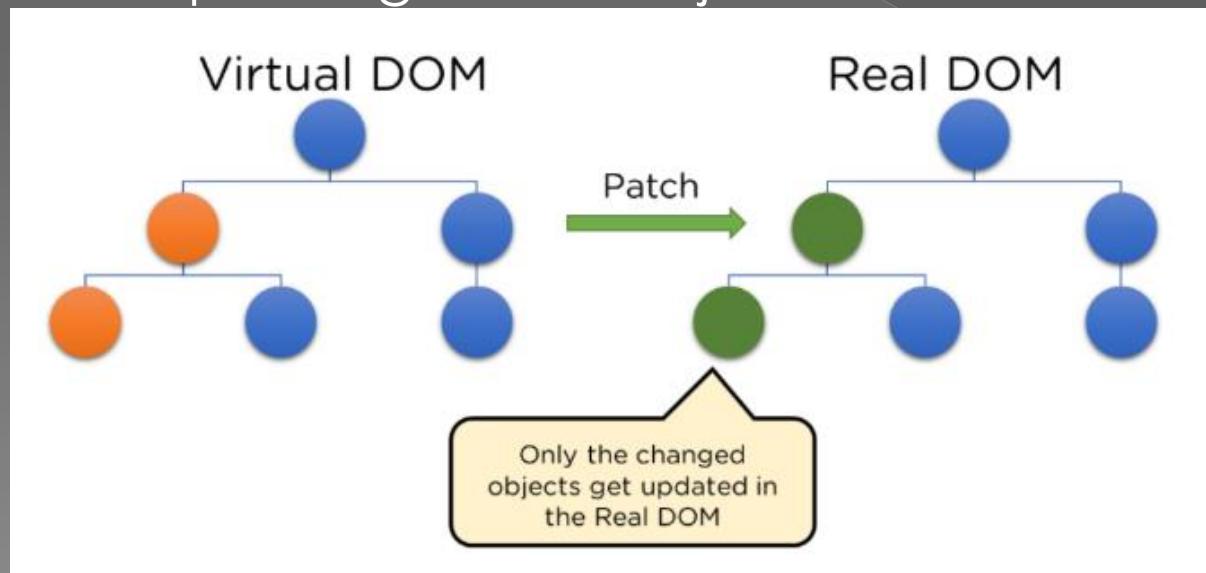


# Virtual DOM

React keeps a lightweight representation of the Real DOM in the memory, and that is known as the Virtual DOM

Manipulating Real DOM is much slower than manipulating virtual DOM, because nothing gets drawn onscreen.

When the state of an object changes, virtual DOM changes only that object in the Real DOM instead of updating all the objects.



# Performance



React uses Virtual DOM that makes the web apps fast.

Complex User Interface is broken down into individual components allowing multiple users to work on each component simultaneously.

# Extension



- React has many extensions that we can use to create full-fledged UI applications
- It provides server-side rendering.
- Supports mobile app development.
- Extended with Flux, Redux, React Native, etc. which helps us to create good-looking UI.

# Debugging



- React applications are extremely easy to test due to a large developer community.
- Facebook even provides a small browser extension that makes React debugging faster and easier

# Component

**Components are the building blocks that comprise a React application representing a part of the user interface.**

**A component used in one area of the application can be reused in another area. This helps speed up the development process.**

**A component can contain several other components.**

# Advantages

- Easy creation of Dynamic web applications
- Performance enhancements
- Reusable components
- Declarative – Tell React what you want and React will build the actual UI
- Unidirectional Data Flow (one way data binding)
- Seamlessly Integrate react into any of your applications. Portion of your webpage or a complete page or even an entire application itself.
- React Native for mobile Applications.

# Limitation

- React Technology accelerates so fast so that it cannot make proper documentation of the project. So, the developer tries to write the instruction on its own.
- React focus on the view part of MVC i.e. UI of the web application. So, to overcome these that to focus on other development we need other tooling set technologies.
- React also have a large size library.
- As some of the command or syntax has to be part of JavaScript as CONDITIONAL BLOCKS, which are not allowed in JSX part of the documents.
- HTML attributes names are in a case insensitive. So, we have to ensure this using names.
- In React, some standard functions will not able to use state effectively instead of that we go for the use of ES6.
- Extension of the files can be emitted only to the JavaScript files so that to ensure that every CSS file imported in ReactJS.
- Some developer thinks JSX as a barrier especially new developer because they have a complaint about the complexity in a learning curve.

# Prerequisites

- HTML, CSS and JavaScript Fundamentals
- ES6 Features
- JavaScript – “this keyword”, filter, map and reducer
- ES6 – let & const, arrow function, template literals, default parameter, destructuring assignment, rest and spread operator, promises, async/await and fetch API.

## Software Requirements:-

1. Node and NPM should be installed in your System
2. Text Editor – (Notepad, Notepad++ or any IDE(visual studio code etc))

# create react project

```
npx create-react-app my-app  
cd my-app  
npm start
```

**npx** is a package runner tool that comes with **npm 5.2+**.

**Npm start -> To start your react app**

# Folder Structure

```
▶ node_modules
◀ public
  ★ favicon.ico
  ◀ index.html
  { manifest.json
◀ src
  # App.css
  JS App.js
  JS App.test.js
  # index.css
  JS index.js
  └ logo.svg
  JS registerServiceWorker.js
  .gitignore
  { package-lock.json
  { package.json
  ⓘ README.md
```

src/ - Contains all of our react codebase.

index.js - Base react component.

Package.json – contains dependencies and scripts required for project.

Node\_modules – contains all dependencies Files which you have installed.

Index.html- Only HTML file as we are creating SPA(single Page Application)

# Component

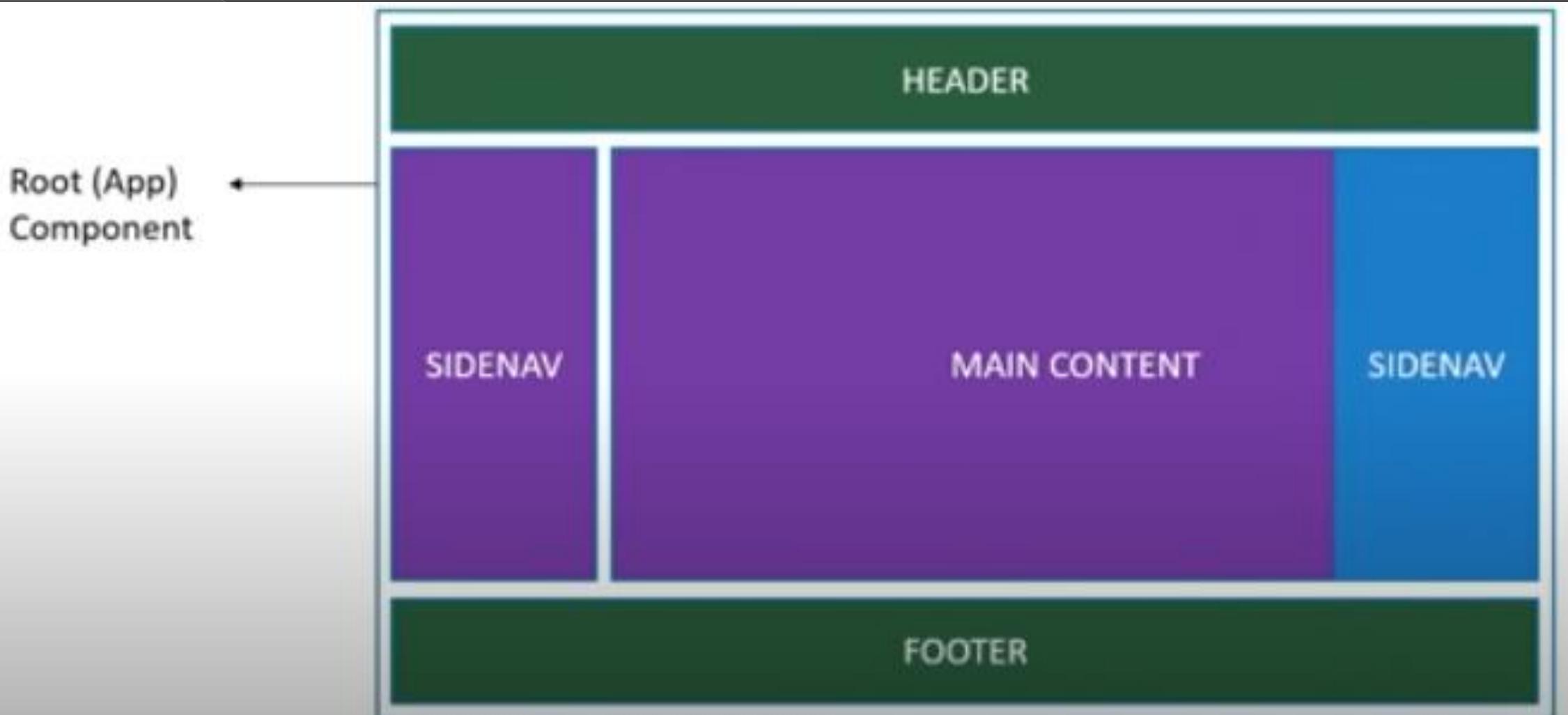
# What is React Component?

- Components are the building blocks of any React app.
- It allow us to split the UI into independent and reusable pieces.

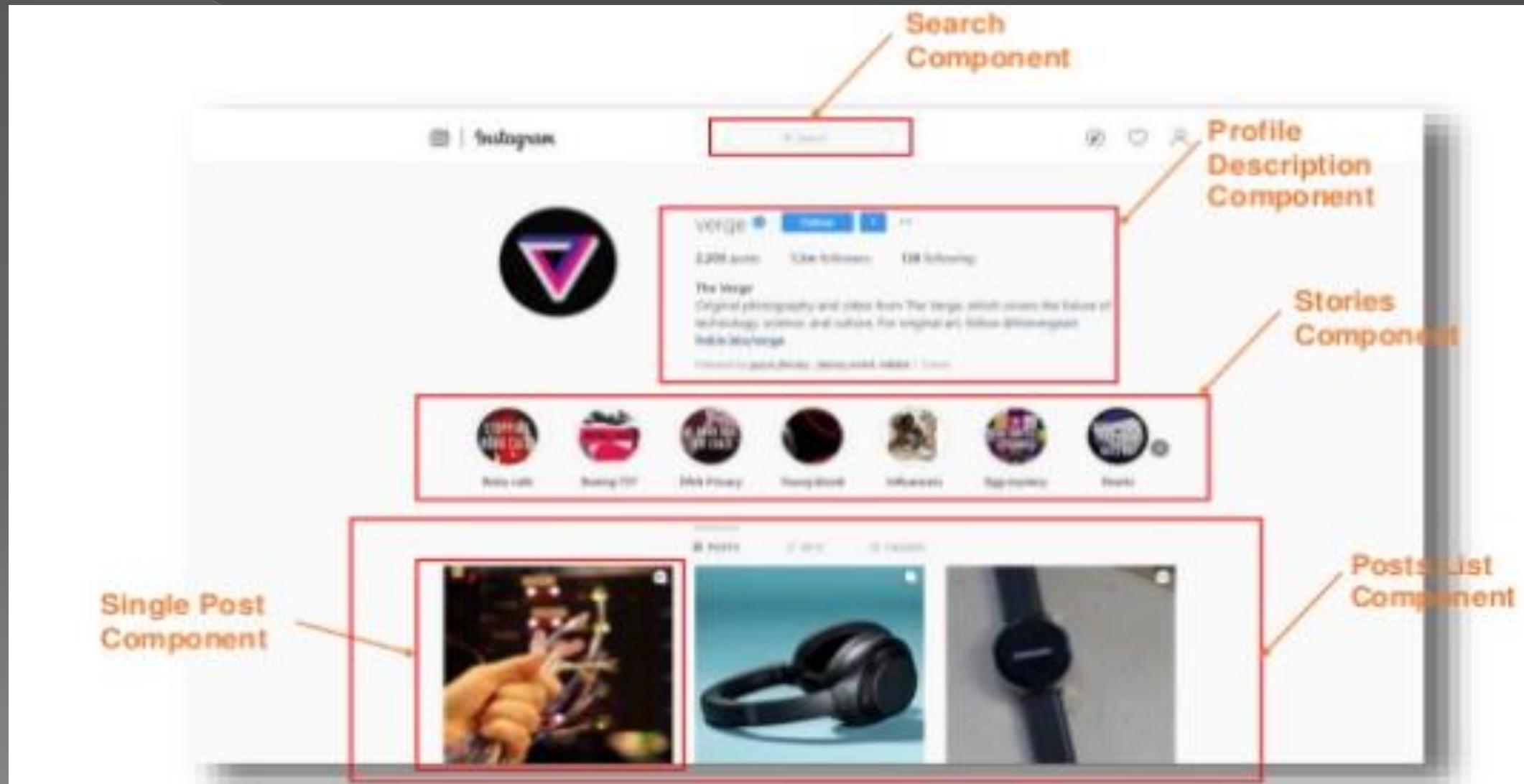
A component is combination of

1. Template using HTML
2. User Interactivity using JS
3. Applying Styles using CSS

**Note: Always start component names with a capital letter. React treats components starting with lowercase letters as DOM tags.**



# Let's see how React works in real time



# Component Types

## Functional Component

JavaScript Functions

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

## Class Component

Class extending Component class

Render method returning HTML

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

# Functional Component



```
Employee.js- const Employee=(data)=> {  
  return <div><p>Name : {data.name}</p>  
    <p>Salary : {data.salary}</p></div>;  
}
```

In App.js -  
<Employee name="Smith" Salary="20000" />

# Class Component



```
class Employee extends Component
{
  render(){
    return <div> <h2>Employee Details...</h2>
      <p> <label>Name : <b>{this.props.Name}</b></label> </p>
      <Department Name={this.props.DeptName}/> </div>;
  }
}
```

# Functional vs. Class Component

Functional	Class
Simple Function	More feature rich
Use Func components as much as possible	Maintain their own private data-state
Absence of 'this' keyword	Complex UI logic
Solution without using state(Uses Hooks)	Provide LifeCycle Methods

# Render()

- Class components uses render function.
- React renders HTML to the web page by using a function called render().
- The purpose of the function is to display the specified HTML code inside the specified HTML element.
- In the render() method, we can read props and state and return our JSX code to the root component of our app.
- In the render() method, we cannot change the state, and we cannot cause side effects( such as making an HTTP request to the webserver).

# Props

- Props is short for properties, that allow us to pass argument or data to components.
- Props are passed to components in the way similar to the HTML tag attributes.
  
- They are used to –
  - Pass custom data to your component
  - Trigger state changes

For Example-

```
Employee.js- const Employee=(props)=> {  
    return <div><p>Name : {props.name}        <p>Salary : {props.salary}}
```

In App.js -

```
<Employee name="Smith" Salary="20000" />
```

# State

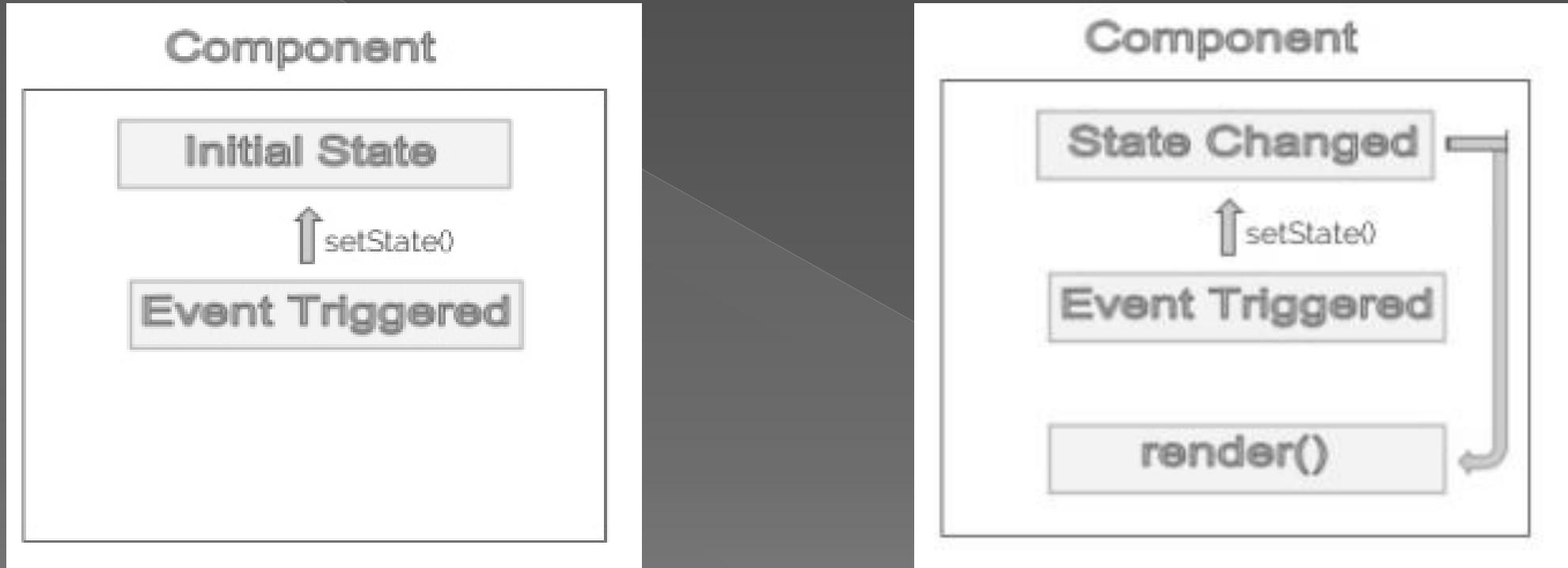
- State of a component is an object that holds some data that may change throughout the component lifecycle..
- We define initial state and then we just have to notify that the state is changed and the react will automatically Render those changes on the Front end behind the scenes.

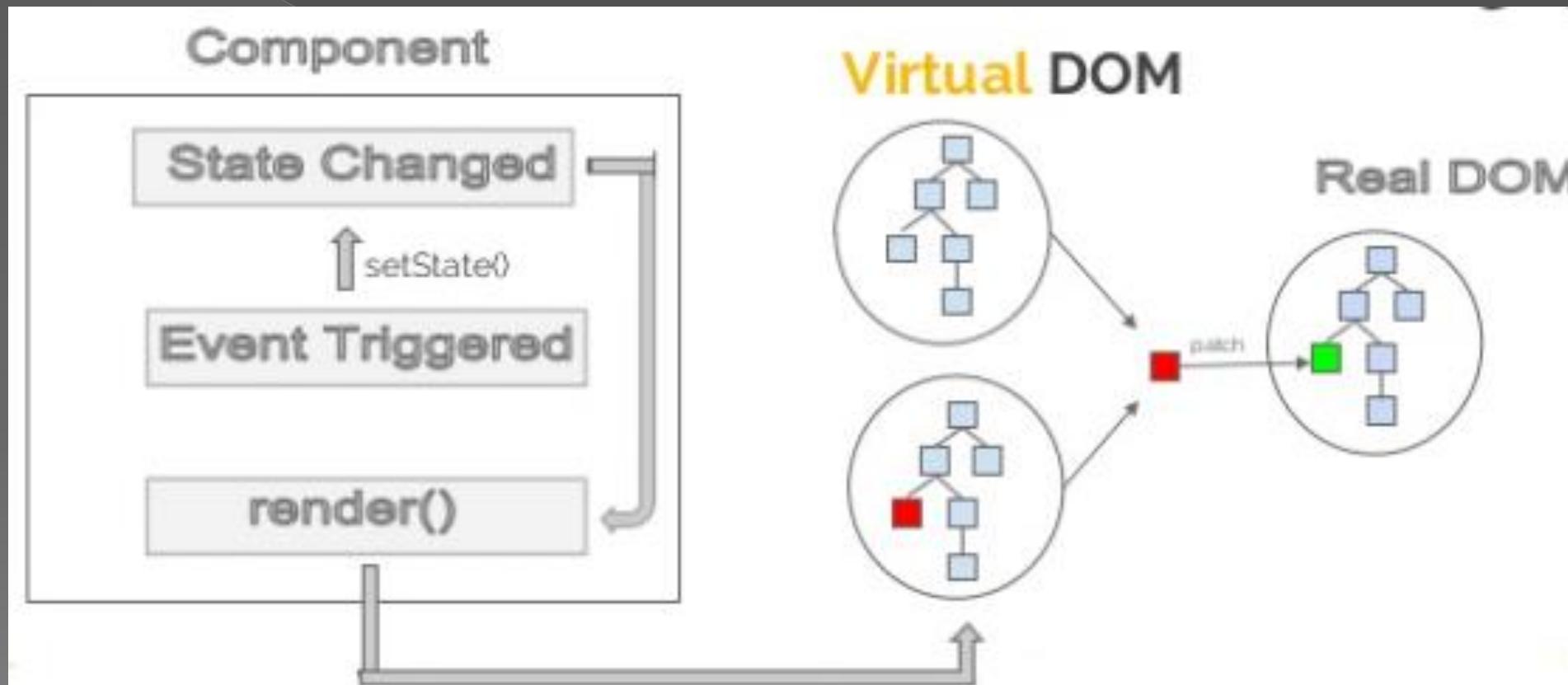
```
import React,{ Component } from 'react';
class App extends Component {
  constructor(props) {
    super(props)
    this.state = {
      Id:101,
      Name:"Ram"
    }
  }
  render(){
    return (
      <div className="App">
        <h2>{this.state.Id}</h2>
        <h2>{this.state.Name}</h2>
      </div>
    );
  }
}
export default App;
```

# Props vs. state

props	state
Props get passed to the component	State is managed within the component
Functional parameters	Variable declared in the function body
Props are immutable	States can be changed
props – functional components this.props – class components	useState Hook – Functional Components This.state – Class Component

# setState Function





# Component Life Cycle Methods

# Life Cycle Phases

Mounting

When an instance of a component is being created and inserted into the DOM

Updating

When a component is being re-rendered as a result of changes to either its props or state

Unmounting

When a component is being removed from the DOM

Error Handling

When there is an error during rendering, in a lifecycle method, or in the constructor of any child component

# Life Cycle Methods

## Mounting

*constructor, static getDerivedStateFromProps, render and componentDidMount*

## Updating

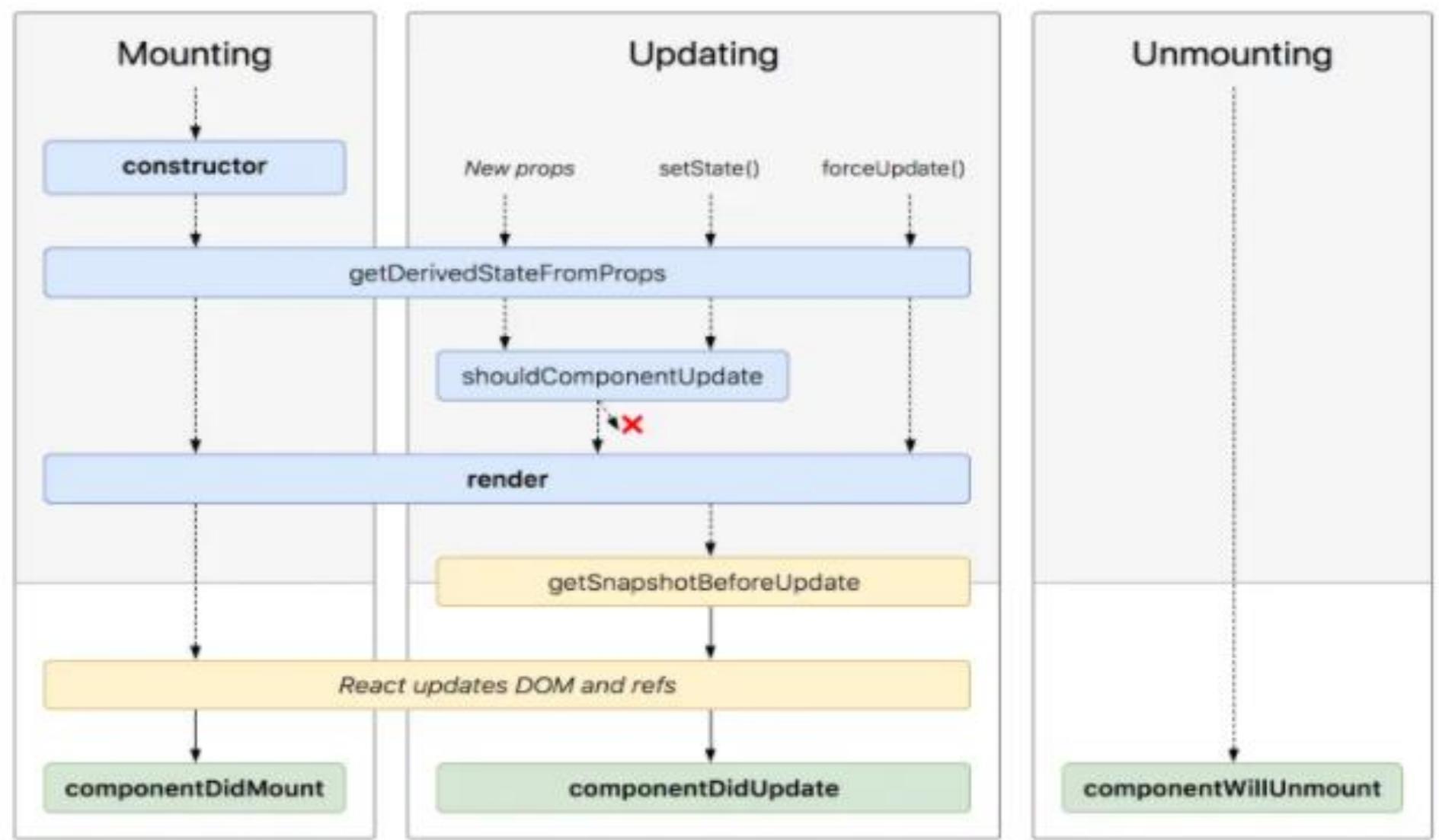
*static getDerivedStateFromProps, shouldComponentUpdate, render,  
getSnapshotBeforeUpdate and componentDidUpdate*

## Unmounting

*componentWillUnmount*

## Error Handling

*static getDerivedStateFromError and componentDidCatch*



# Mounting Life Cycle Phases

`constructor( props )`

A special function that will get called whenever a new component is created.

Initializing state  
Binding the event handlers

Do not cause side effects. Ex: HTTP requests

`super(props)`  
Directly overwrite `this.state`

# Mounting Life Cycle Phases

constructor( props )



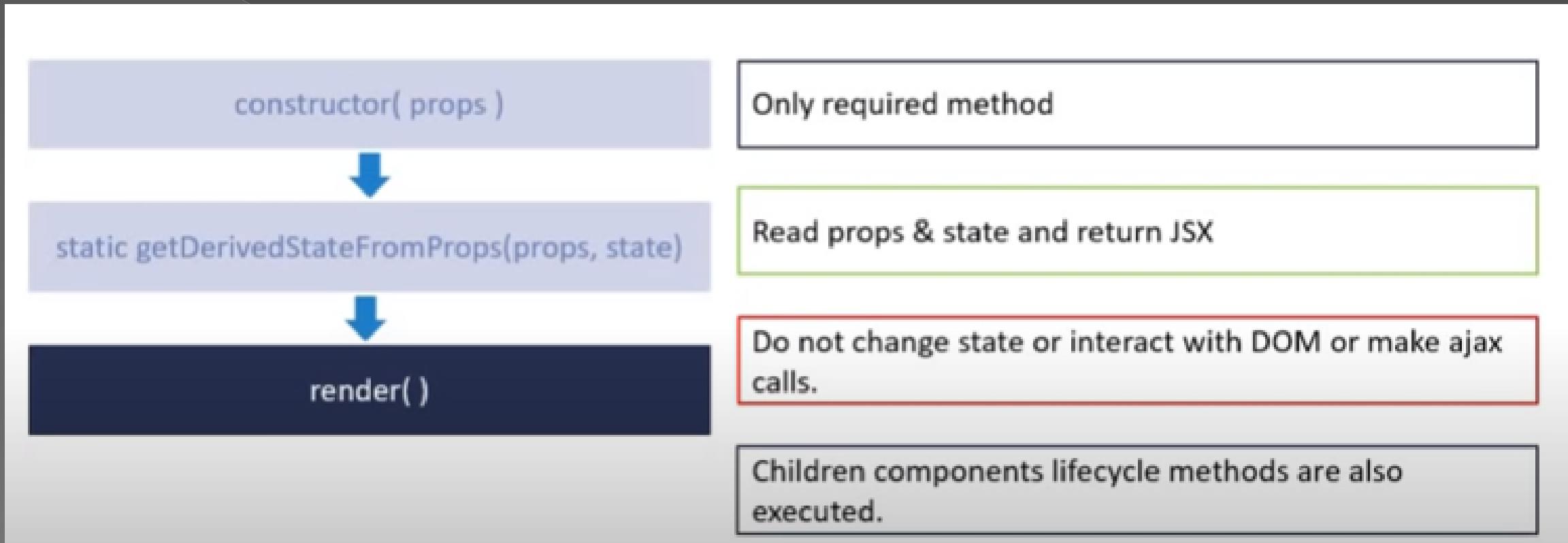
static getDerivedStateFromProps( props, state )

When the state of the component depends on changes in props over time.

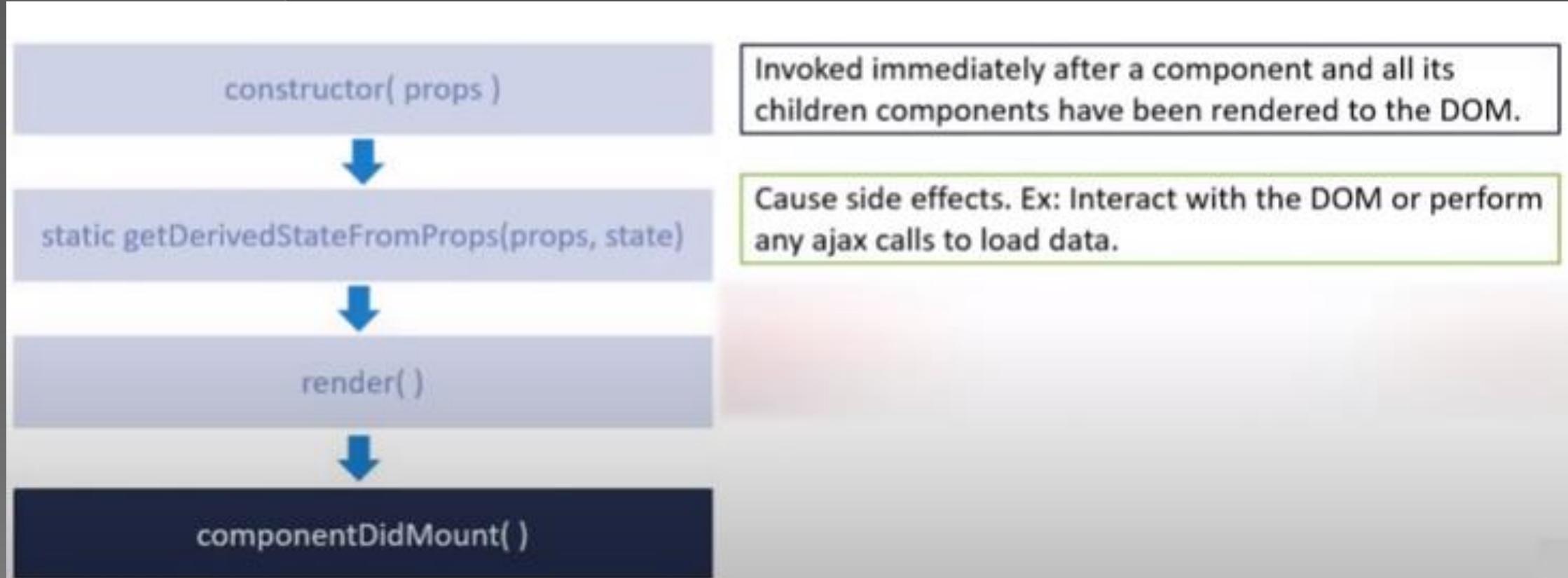
Set the state

Do not cause side effects. Ex: HTTP requests

# Mounting Life Cycle Phases



# Mounting Life Cycle Phases



# Updating Life Cycle Phases

```
static getDerivedStateFromProps( props, state)
```

Method is called every time a component is re-rendered

Set the state

Do not cause side effects. Ex: HTTP requests

# Updating Life Cycle Phases

```
static getDerivedStateFromProps( props, state)
```



```
shouldComponentUpdate( nextProps, nextState)
```

Dictates if the component should re-render or not

Performance optimization

Do not cause side effects. Ex: HTTP requests  
Calling the setState method

# Updating Life Cycle Phases

```
static getDerivedStateFromProps( props, state)
```

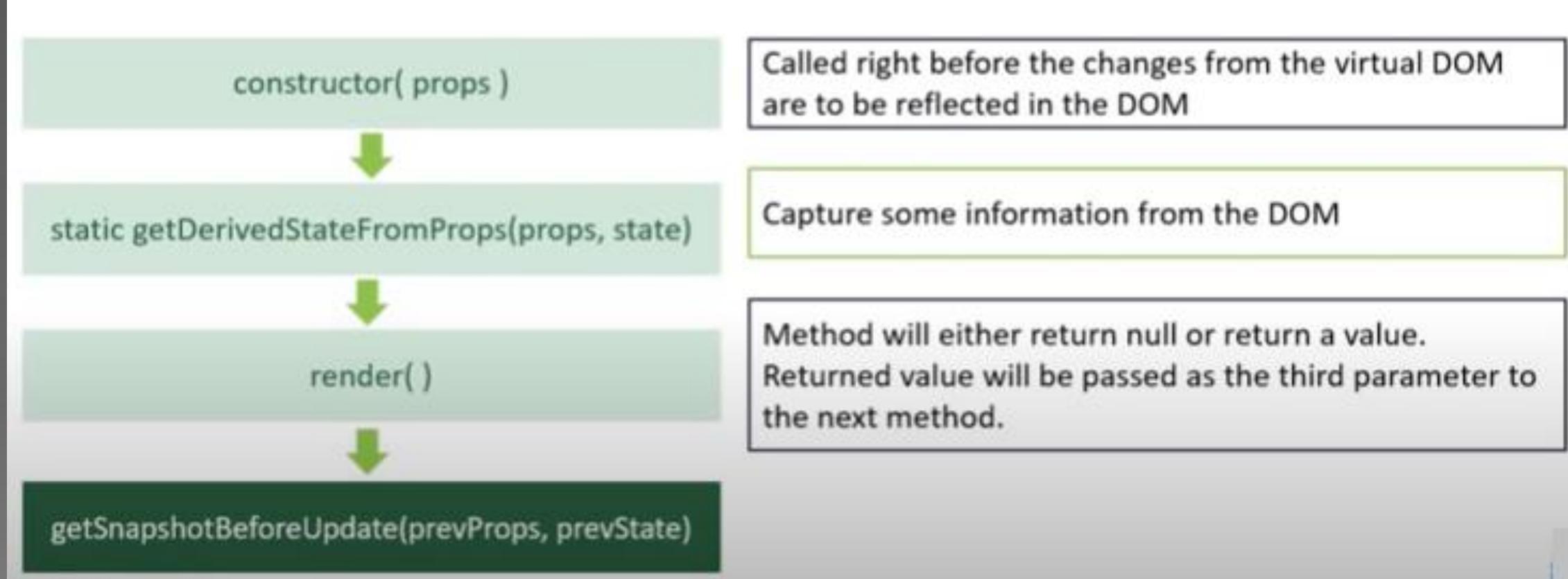
Only required method

```
shouldComponentUpdate( nextProps, nextState)
```

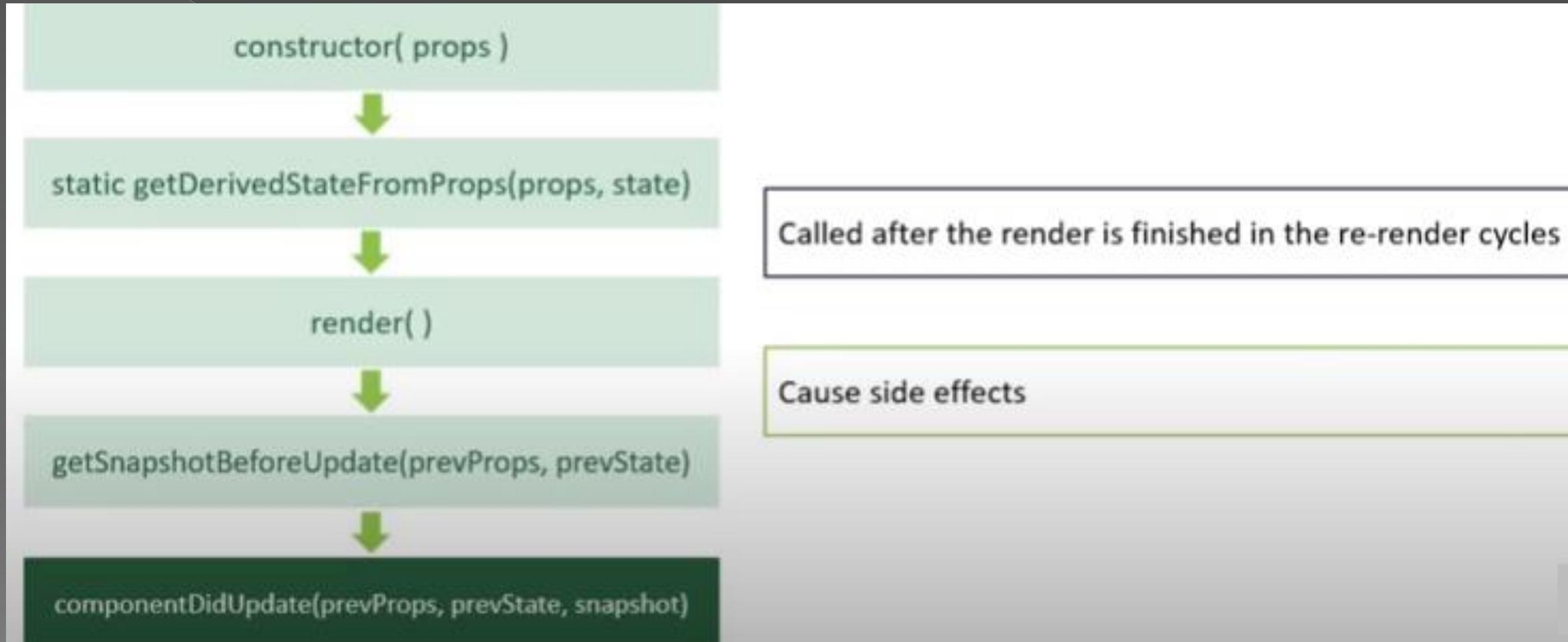
Read props & state and return JSX

```
render( )
```

# Updating Life Cycle Phases



# Updating Life Cycle Phases



# Unmounting Life Cycle Phases

`componentWillUnmount( )`

Method is invoked immediately before a component is unmounted and destroyed.

Cancelling any network requests, removing event handlers, cancelling any subscriptions and also invalidating timers.

Do not call the `setState` method.

# Error Handling Life Cycle Phases

```
static getDerivedStateFromError(error)
```

```
componentDidCatch(error, info)
```

When there is an error either during rendering, in a lifecycle method, or in the constructor of any child component.

# Event Handling

- Just like HTML, React can perform actions based on user events.
- React has the same events as HTML: click, change, mouseover etc.
- React events are written in camelCase syntax.
- React event handlers are written inside curly braces.
- For methods in React, the this keyword should represent the component that owns the method.
- That is why you should use arrow functions. With arrow functions, this will always represent the object that defined the arrow function.

```
class App extends Component {  
  clickHandler = () => { alert("Button Clicked"); }  
  render() {  
    return (  
      <button onClick={this.clickHandler}>Click Me</button>  
    );  
  }  
}  
  
export default App
```

# Event Handling

## Passing Arguments : -

- **Make an anonymous arrow function**

```
<button onClick={() => this.clickHandler("Hello")}>  
Click Me!!</button>
```

- **Bind the event handler to this**

```
<button onClick={this.clickHandler.bind(this, "Hello")}>Click  
Me!!</button>
```

# **Styling with CSS Basics**

- **CSS Stylesheets**
- **Inline Styling**
- **CSS Modules**
- **CSS in JS Libraries (Styled Component)**

# Inline Stylesheet

```
<h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
```

```
render() {
  const mystyle = {
    color: "white",
    backgroundColor: "DodgerBlue",
    padding: "10px",
    fontFamily: "Arial"
  };
  return (
    <div>
      <h1 style={mystyle}>Hello Style!</h1>
      <p>Add a little style!</p>
    </div>
  );
}
```

# CSS Modules

The CSS inside a module is available only for the component that imported it.

Create the CSS module with the .module.css.

For Example : -

**mystyle.module.css**

```
.bigblue {  
  color: DodgerBlue;  
  padding: 40px;  
  font-family: Arial;  
  text-align: center;  
}
```

**App.js**

```
import styles from './mystyle.module.css';  
  
render() {  
  return <h1 className={styles.bigblue}>Hello Car!</h1>;  
}
```

# Conditional Rendering

Conditional rendering is a term to describe the ability to render different user interface (UI) markup if a condition is true or false. In React, it allows us to render different elements or components based on a condition. This concept is applied often in the following scenarios:

- Rendering external data from an API.
- Showing or hiding elements.
- Toggling application functionality.
- Implementing permission levels.
- Handling authentication and authorization.

## Conditional Rendering Approaches:

1. If/else
2. Element Variables
3. Ternary conditional operator
4. Short Circuit Operator

# If/else Approaches

```
class App extends Component {  
    // ...  
    render() {  
        let {isLoggedIn} = this.state;  
        if (isLoggedIn) {  
            return (  
                <div className="App">  
                    <button>Logout</button>  
                </div>  
            );  
        } else {  
            return (  
                <div className="App">  
                    <button>Login</button>  
                </div>  
            );  
        }  
    }  
}
```

# Element Variables

```
class App extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      isLoggedIn: true  
    };  
  }  
  render() {  
    let { isLoggedIn } = this.state;  
    let AuthButton;  
    if (isLoggedIn) {  
      AuthButton = <button>Logout</button>;  
    } else {  
      AuthButton = <button>Login</button>;  
    }  
    return (  
      <div className="App">  
        {AuthButton}  
      </div>  
    );  
  }  
}  
export default App;
```

# Using Ternary Operators

```
class App extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      isLoggedIn: true  
    };  
  }  
  render() {  
    let { isLoggedIn } = this.state;  
    return (  
      <div className="App">  
        {isLoggedIn ? <button>Logout</button> : <button>Login</button>}  
      </div>  
    );  
  }  
}  
export default App;
```

# Using Logical && (Short Circuit Evaluation)

Short circuit evaluation is a technique used to ensure that there are no side effects during the evaluation of operands in an expression. The logical `&&` helps you specify that an action should be taken only on one condition, otherwise, it would be ignored entirely.

```
class App extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      isLoggedIn: true  
    };  
  }  
  render() {  
    let { isLoggedIn } = this.state;  
    return (  
      <div className="App">  
        {isLoggedIn && <button>Logout</button>}  
      </div>  
    );  
  }  
}  
export default App;
```

# List Rendering

Using array index:

```
const employee = ['zbc','xyz'];
return(
  <div>
    <h2>{employee[0]}</h2>
    <h2>{employee[1]}</h2>
  </div>
)
```

Array.map():

```
const employee = ['abc','xyz'];
return(
  <div> {employee.map(emp => <h2>{emp}</h2>)} </div>
)
```

# List Rendering

Now we will see example of having an array with key-value pair in each array having multiple values:

```
function EmployeeList(){
    const employee = [
        {
            name:'abc',
            salary:'50$',
            position:'Jr. Developer'
        },
        {
            name:'xyz',
            salary:'100$',
            position:'Sr. Developer'
        },
        {
            name:'mno',
            salary:'150$',
            position:'Project Manager'
        }
    ];
    const employeeList = employee.map(emp => <h2>My name is {emp.name} working as {emp.position} and having salary {emp.salary}</h2>);
    return(
        <div>
            {employeeList}
        </div>
    )
}
export default EmployeeList;
```

# Render List in Sub-Component

```
import Employees from './Employees';
function EmployeeList(){
    const employee = [
        {
            name:'abc',
            salary:'50$',
            position:'Jr. Developer'
        },
        {
            name:'xyz',
            salary:'100$',
            position:'Sr. Developer'
        },
        {
            name:'mno',
            salary:'150$',
            position:'Project Manager'
        }
    ];
    const employeeList = employee.map(emp =>
        <Employees emp={emp}></Employees>
    );
    return <div>{employeeList}</div>;
}
export default EmployeeList;
```

```
function Employees({emp}){
    return(
        <div>
            <h2>My name is {emp.name} working as
                {emp.position} and having salary
                {emp.salary} </h2>
        </div>
    )
}
export default Employees;
```

# List and Key Props

When we fetch list in sub-component then we will find there are errors related to keys in console. It will show a warning related to keys as each child in a list should have a unique key prop. This error can be resolved by defining the key to each list item generated using JSX. The key defined should not be the same for any list item.

```
const employeeList = employee.map(emp => <Employees key= {emp.id} emp={emp}></Employees>);  
  return <div>{employeeList}</div>;  
}
```

The important point that needs to be kept in mind when using key prop is that key prop cannot be used in child component.

## Importance of key prop:

- Key prop helps to easily identify which item is added, updated or removed.
- Key prop helps in updating user interface efficiently.
- Keys provide stable identity to elements.

# Fragment

We know that we make use of the render method inside a component whenever we want to render something to the screen. We may render a single element or multiple elements, though rendering multiple elements will require a 'div' tag around the content as the render method will only render a single root node inside it at a time.

when we are trying to render more than one root element we have to put the entire content inside the 'div' tag which is not loved by many developers. So in React 16.2 version, **Fragments** were introduced, and we use them instead of the extraneous 'div' tag.

Syntax:

```
<React.Fragment>
  <h2>Child-1</h2>
  <p> Child-2</p>
</React.Fragment>
```

## Shorthand Fragment :

```
<>
  <h2>Child-1</h2>
  <p> Child-2</p>
</>
```

# React Form Handling

# Creating Form

React offers a stateful, reactive approach to build a form. The component rather than the DOM usually handles the React form. In React, the form is usually implemented by using controlled components. There are mainly two types of form input in React.

- Uncontrolled component: Where form data is handled by the DOM itself. We will use ref to get the input values and Perform Operations using this data.
- Controlled component: An input form element whose value is controlled by React in this way is called a “controlled input or Controlled Component”.

# Controlled Components

```
this.state = {  
    email: ''  
}  
  
this.changeEmailHandler = (event) => {  
    this.setState({email: event.target.value})  
}  
  
<input type='text' value={this.state.email} onChange={this.changeEmailHandler} />
```

```
import React, { Component } from 'react'
class Form extends Component {
    constructor(props) {
        super(props)
        this.state = {
            username: '',
        }
    }
    handleUsernameChange = event => {
        this.setState({
            username: event.target.value
        })
    }
    handleSubmit = event => {
        alert(` ${this.state.username}`)
        event.preventDefault()
    }
    render() {
        return (
            <form onSubmit={this.handleSubmit}>
                <div>
                    <label>Username </label>
                    <input type="text" value={this.state.username} onChange={this.handleUsernameChange}>
                </div>
                <button type="submit">Submit</button>
            </form>
        )
    }
}
export default Form
```

# Uncontrolled Components

To write an uncontrolled component, you need to use a ref to get form values from the DOM.

In other words, there is no need to write an event handler for every state update. You can use a ref to access the input field value of the form from the DOM.

```
import React, { Component } from 'react';
class App extends Component {
  constructor(props) {
    super(props);
    this.updateSubmit = this.updateSubmit.bind(this);
    this.input = React.createRef();
  }
  updateSubmit(event) {
    alert('You have entered the UserName and Age successfully.');
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.updateSubmit}>
        <h1>Uncontrolled Form Example</h1>
        Name: <input type="text" ref={this.input} />
        Age: <input type="text" ref={this.input} />
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
export default App;
```

# Controlled vs uncontrolled component

Controlled	Uncontrolled
It does not maintain its internal state.	It maintains its internal states.
Here, data is controlled by the parent component.	Here, data is controlled by the DOM itself.
It accepts its current value as a prop.	It uses a ref for their current values.
It allows validation control.	It does not allow validation control.
It has better control over the form elements and data.	It has limited control over the form elements and data.

# Refs

**Refs is the shorthand used for references in React. It is similar to keys in React. It is an attribute which makes it possible to store a reference to particular DOM nodes or React elements.**

## When to Use Refs:

- Refs can be used in the following cases:
- When we need DOM measurements such as managing focus, text selection, or media playback.
- It is used in triggering imperative animations.
- When integrating with third-party DOM libraries.
- It can also use as in callbacks.

## When to not use Refs:

- Its use should be avoided for anything that can be done declaratively. For example, instead of using open() and close() methods on a Dialog component, you need to pass an isOpen prop to it.
- You should have to avoid overuse of the Refs.

# How to create Refs

Refs can be created by using `React.createRef()`.

**How to access Refs:**

```
const node = this.callRef.current;
```

**Refs current Properties:**

- The ref value differs depending on the type of the node:
- When the ref attribute is used in HTML element, the ref created with `React.createRef()` receives the underlying DOM element as its `current` property.
- If the ref attribute is used on a custom class component, then ref object receives the mounted instance of the component as its `current` property.
- The ref attribute cannot be used on function components because they don't have instances.

# Callback refs

it gives more control when the refs are set and unset. Instead of creating refs by `createRef()` method.

```
<input type="text" ref={element => this.callRefInput = element} />
```

It can be accessed as below:

```
this.callRefInput.value
```

# Forwarding Ref from one component to another component

```
import React, { Component } from 'react';
import { render } from 'react-dom';

const TextInput = React.forwardRef((props, ref) => (
  <input type="text" placeholder="Hello World" ref={ref} />
));

const inputRef = React.createRef();

class CustomTextInput extends Component {
  handleSubmit = e => {
    e.preventDefault();
    console.log(inputRef.current.value);
  };
  render() {
    return (
      <div>
        <form onSubmit={e => this.handleSubmit(e)}>
          <TextInput ref={inputRef} />
          <button>Submit</button>
        </form>
      </div>
    );
  }
}
```

# Error Boundary

A class component that implements either one or both the life cycle methods `getDerivedStateFromError` or `componentDidCatch` becomes an error boundary.

The static method `getDerivedStateFromError` method is used to render a fallback UI after an error is thrown and the `componentDidCatch` method is used to log the error information.

Error boundaries do not catch errors inside event handlers.

Error boundaries do not catch errors also for:

- Any Asynchronous code we write (e.g. `setTimeout` )
- For Server side rendering code
- For Errors thrown in the error boundary component class itself (rather than its children)

```

import React, { Component } from 'react'
export class ErrorBoundary extends Component {
  constructor(props) {
    super(props)
    this.state = {
      hasError: false
    }
  }
  static getDerivedStateFromError(error) {
    return { hasError: true }
  }
  componentDidCatch(error, info) {
    console.log(error)
    console.log(info)
  }
  render() {
    if (this.state.hasError) {
      return <h1>Something went wrong.</h1>
    }
    return this.props.children
  }
}
export default ErrorBoundary

```

```

import React from 'react'
function Hero ({ heroName }) {
  if (heroName === 'Joker') {
    throw new Error(' Not a hero!')
  }
  return <h1>{heroName}</h1>
}
export default Hero

```

```

class App extends Component {
  render() {
    return (
      <div className="App">
        <Hero heroName="Batman" />
        <Hero heroName="Superman" />
        <ErrorBoundary>
          <Hero heroName="Joker" />
        </ErrorBoundary>
      </div>
    )
  }
}

```

# Context

**Context allows passing data through the component tree without passing props down manually at every level.**

**In React application, we passed data in a top-down approach via props. Sometimes it is inconvenient for certain types of props that are required by many components in the React application. Context provides a way to pass values between components without explicitly passing a prop through every level of the component tree.**

**There are three main steps to use the React context into the React application:**

- **Create context**
- **Setup a context provider and define the data which you want to store.**
- **Use a context consumer whenever you need the data from the store**

# React Context API

The React Context API is a component structure, which allows us to share data across all levels of the application. The main aim of Context API is to solve the problem of prop drilling (also called "Threading"). The Context API in React are given below.

- **React.createContext :**

```
const MyContext = React.createContext(defaultValue);
```

- **Context.provider**

```
<MyContext.Provider value={/* some value */}>
```

- **Context.Consumer**

```
<MyContext.Consumer>  
  {value => /* render something which is based on the context value */}  
</MyContext.Consumer>
```

- **Class.contextType**

## UserContext.js

```
import React from 'react'

const UserContext = React.createContext('default')

const UserProvider = UserContext.Provider
const UserConsumer = UserContext.Consumer

export { UserProvider, UserConsumer }
```

## Columns.js

```
import React, { Component } from 'react'
import UserConsumer from './UserContext'
export default class ColumnFragment extends Component {
  render(){
    return (
      <>
        <UserConsumer>
          {(uname)=><div>Hello {uname}</div>}
        </UserConsumer>
      </>
    )
  }
}
```

```
import { UserProvider } from './Components/UserContext';
class App extends Component{
  render(){
    return (
      <div>
        <UserProvider value="Infoway">
          <TableFragment/>
        </UserProvider>
      </div>
    );
  }
}
export default App;
```

# Class.contextType

UserContext.js

```
import React from 'react'
const UserContext=React.createContext();
const UserProvider=UserContext.Provider
const UserConsumer=UserContext.Consumer
export {UserProvider, UserConsumer}
export default UserContext;
```

```
import React, { Component } from 'react'
import UserContext from './UserContext'
export default class ColumnFragment extends Component {
  static contextType=UserContext;
  render(){
    return (
      <>
        {this.context.Id}{this.context.Name}
      </>
    )
  }
// ColumnFragment.contextType=UserContext;
```

```
import { UserProvider } from './Components/UserContext';
class App extends Component{
  constructor(props) {
    super(props)
    this.state = {
      data:{Id:101,Name:"Ram"}
    }
  }
  changedata=()=>{
    this.setState({
      data:{Id:102,Name:"Seema"}
    })
  }
  render(){
    return (
      <div>
        <UserProvider value={this.state.data}>
          <TableFragment/>
        </UserProvider>
        <button onClick={this.changedata}>Change Data of Context</button>
      </div>
    );
  }
}
export default App;
```

# END

# Thank you !!

# **Web Programming Technologies**

Harshita Maheshwari

# Session-6

# Topics to be covered.....

- What is Responsive Web Design?
- Responsive web design-Frameworks
- Overview and Need to use Bootstrap
- Bootstrap Grid System
- Bootstrap Typography
- Bootstrap tables
- Images
- Alerts
- Badge
- List Groups
- Buttons and Button Groups
- Bootstrap Cards
- Dropdowns
- Collapse
- Modal

# Responsive Web Designing (RWD)

Responsive Web Designing (RWD) is a process of designing a single website to be used and compatible on different portable or handy electronic devices.

Also known as **Adaptive Web Designing (AWD)**

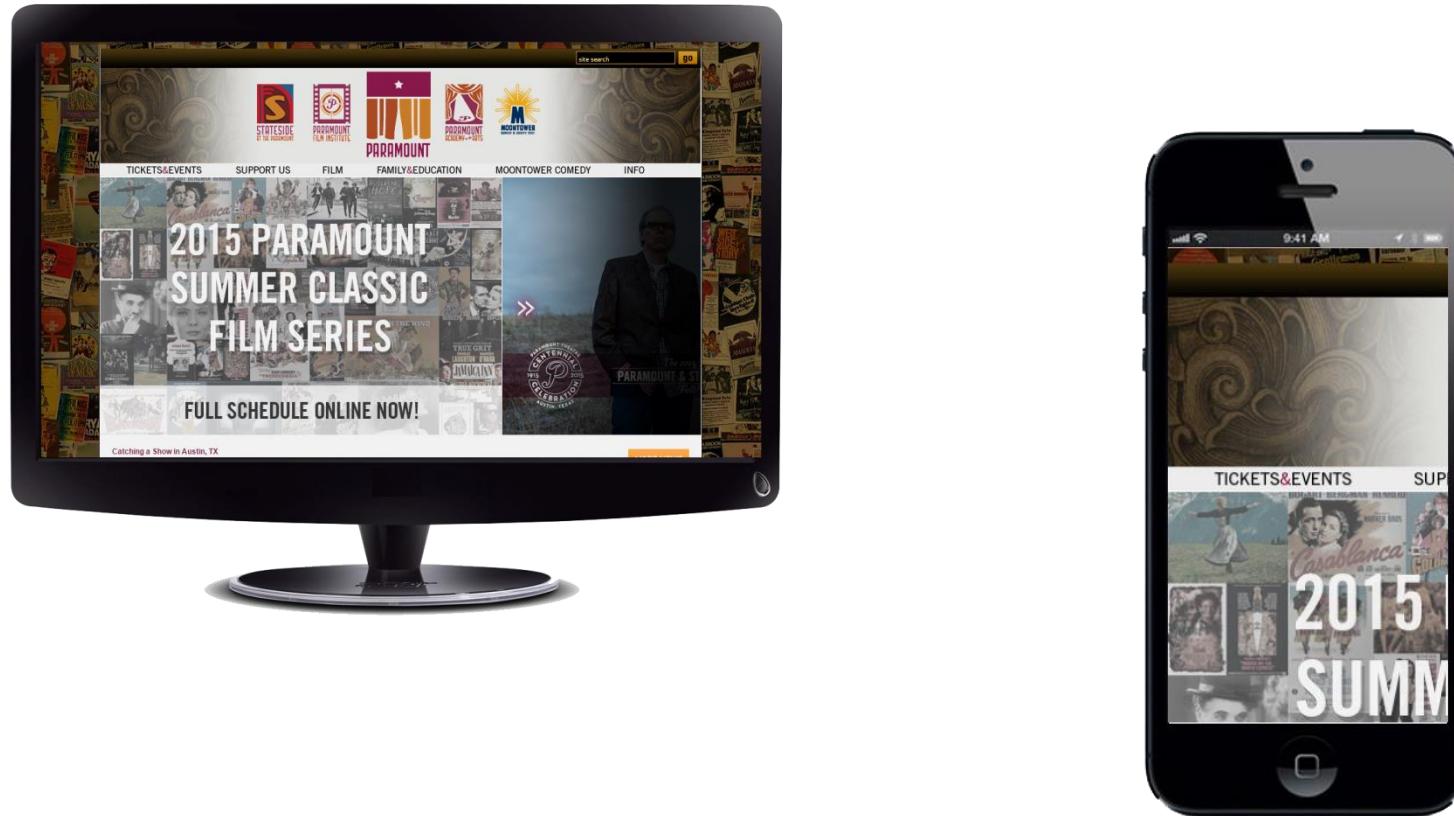
It regarded as an integrated approach of designing through which compelling and easy to use websites are built, to give an optimal viewing user experience across a wide variety of devices starting from desktop computers to mobile phones

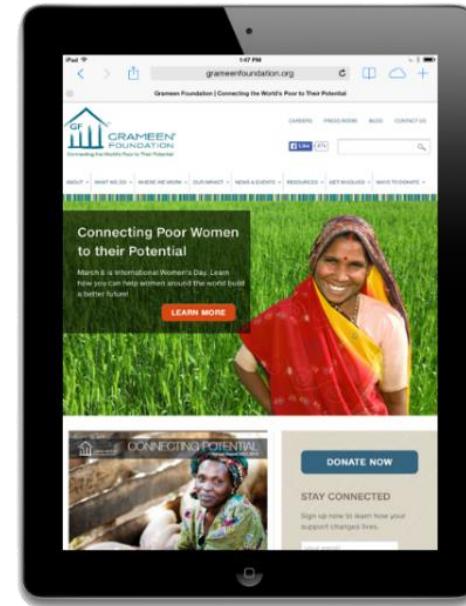
**Responsive Websites Offers:**

- ✓ Smooth navigation
- ✓ Easy reading
- ✓ Minimum pinching
- ✓ Reduces scrolling and zooming.
- ✓ Excellent user experience.

# State of today's Web







# Need of Responsive Web Design

- Increasing Demand for Smartphones
- Multiple Screen Sizes and Mobile Browsers
- Wide Usage of Internet
- Permits wider browser support
- Compulsory for Getting Good Business

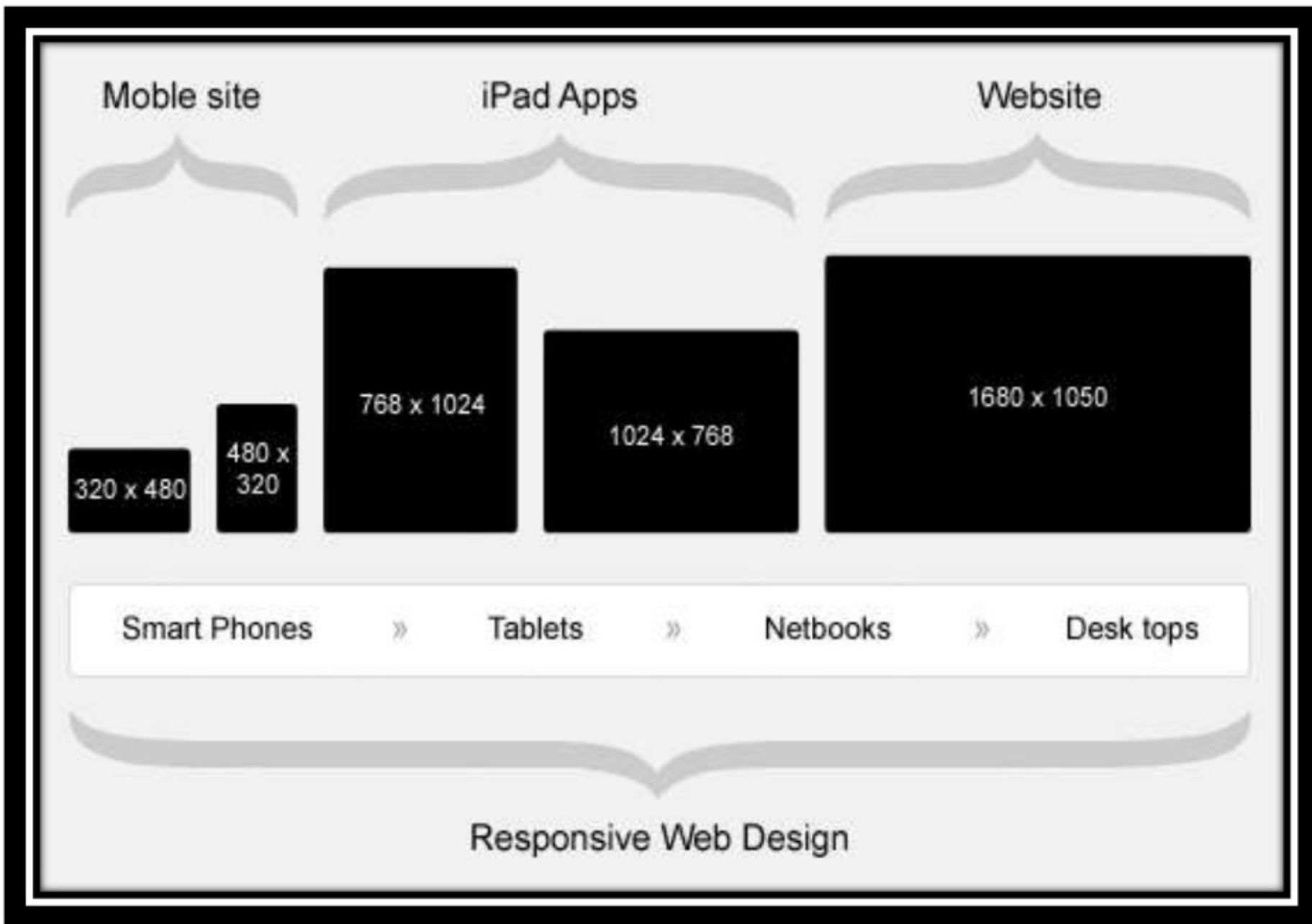
## CONTENT IS LIKE WATER



“ You put water into a cup it becomes **the cup**.  
You put water into a bottle it becomes **the bottle**.  
You put it in a teapot, it becomes **the teapot**. ”

Josh Clark (*originally Bruce Lee*) - Seven deadly mobile myths

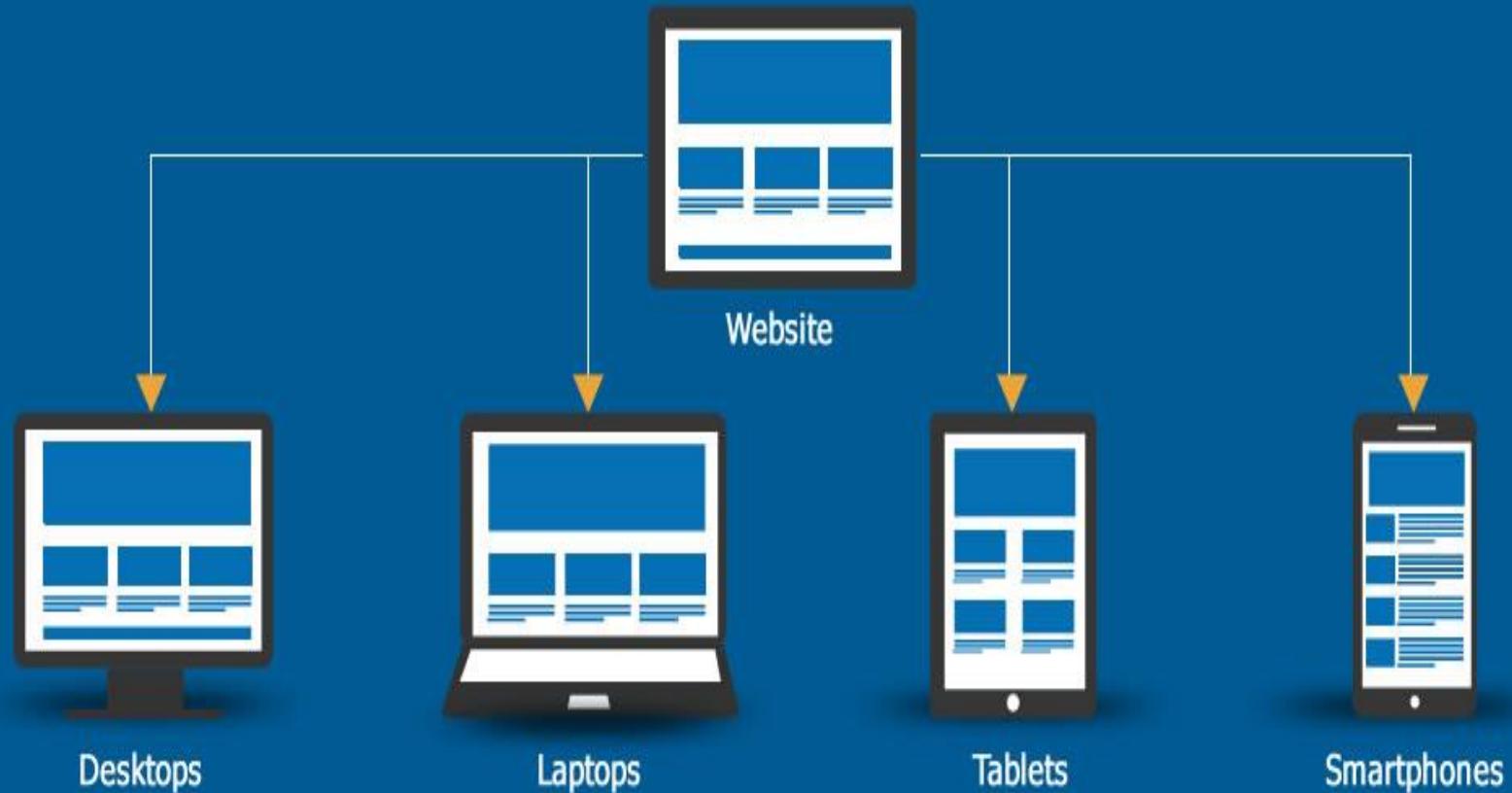
Illustration by Stéphanie Walter



## What is Responsive Web Design ?



One URL      Single content      Same Code      Using CSS3 Media Queries      Multi screen



A freely available design framework for websites  
and web applications

Based upon HTML5, CSS and JavaScript

Released on GitHub in August 2011

Developed by Mark Otto and Jacob Thornton at  
Twitter.



Bootstrap is the most popular framework for RAPID  
WEBSITE PROTOTYPING..

# Why use Bootstrap

## ➤ Mobile first approach:

Since Bootstrap 3, the framework consists of Mobile first styles throughout the entire library instead of in separate files.

## ➤ Browser Support:

It is supported by all popular browsers.

## ➤ Easy to get started:

With just the knowledge of HTML and CSS anyone can get started with Bootstrap. Also the Bootstrap official site has a good documentation.

## ➤ Responsive design:

Bootstrap's responsive CSS adjusts to Desktops, Tablets and Mobiles.

- **jQuery isn't required anymore** (however you can still use jQuery methods if you wish)
- Dropped support for **Internet Explorer**
- The use of JavaScript is **minimized in favor of CSS (CSS custom properties)**
- Almost each option **is available as data-attribute** (it can be set manually without JS)
- Enhanced **Grid system**
- Enhanced **modularity**
- Enhanced **customization**
- New components/helpers/utilities/variations
- Some components were removed (i.e. Jumbotron)
- Enhanced **Icons (SVG Icon Library)**

# Where to Get Bootstrap?

There are two ways to start using Bootstrap on your own web site.

- **BootstrapCDN** is a free and public *Content Delivery Network*. Users of BootstrapCDN can load CSS, JavaScript and images remotely, from its servers.

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css"  
rel="stylesheet">
```

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/js/bootstrap.min.js"></script>
```

OR

- We can download the latest version of Bootstrap from <http://getbootstrap.com/>.

# File Structure

Once the compiled version Bootstrap is downloaded, extract the ZIP file, and you will see the following file/directory structure:

```
bootstrap/
├── css/
│   ├── bootstrap-grid.css
│   ├── bootstrap-grid.css.map
│   ├── bootstrap-grid.min.css
│   ├── bootstrap-grid.min.css.map
│   ├── bootstrap-reboot.css
│   ├── bootstrap-reboot.css.map
│   ├── bootstrap-reboot.min.css
│   ├── bootstrap-reboot.min.css.map
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   └── bootstrap.min.css.map
└── js/
    ├── bootstrap.bundle.js
    ├── bootstrap.bundle.js.map
    ├── bootstrap.bundle.min.js
    ├── bootstrap.bundle.min.js.map
    ├── bootstrap.js
    ├── bootstrap.js.map
    ├── bootstrap.min.js
    └── bootstrap.min.js.map
```

# Bootstrap Grid System

Bootstrap grid system is used for creating page layout through a series of rows and columns.

The grid system consists of 12 columns.

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1				
span 4			span 4				span 4								
span 4				span 8											
span 6						span 6									
span 12															

## Bootstrap Grid Classes:

Device	Class
Extra small (<576px)	.col-
Small ( $\geq$ 576px)	.col-sm-
Medium ( $\geq$ 768px)	.col-md-
Large( $\geq$ 992px)	.col-lg-
Extra large ( $\geq$ 1200px)	.col-xl-
Xxl ( $\geq$ 1400px)	.col-xxl-

```
<div class="container">  
  <div class="row">  
    <div class="col-*-*"></div>  
    <div class="col-*-*"></div>  
  </div>  
  <div class="row">...</div>  
</div>  
<div class="container">....
```

# Bootstrap Grid System Example: Medium and Extra Small Device

Here we had used 2 div's and gave them the 50%/50% split at the medium and extra small viewport width:

```
<div class="container">
  <div class="row">
    <div class="col-md-6 col-4">
      <p>div 1...</p>
      <p>lgrid1 ...</p>
    </div>
    <div class="col-md-6 col-8">
      <p>div2...</p>
      <p>grid2...</p>
    </div>
  </div>
```

Bootstrap requires a containing element to wrap site contents and house our grid system.

Use **.container** for a responsive fixed width container.

Use **.container-fluid** for a full width container, spanning the entire width of your viewport.

```
div class="container">
  ...
</div>

<div class="container-fluid">
  ...
</div>
```

# Bootstrap Grid column Offset

Move columns to the right.

These classes increase the left margin of a column by \* columns.

.col-md-4

.col-md-4 .offset-md-4

.col-md-3 .offset-md-3

.col-md-3 .offset-md-3

.col-md-6 .offset-md-3

Make it easy to create heading, paragraph, ol, ul, inline element etc. in a way that would be appealing to the users.

```
<span class="h1">H1</span>
<span class="h2">H2</span>
<span class="h3">H3</span>
<span class="h4">H4</span>
<span class="h5">H5</span>
<span class="h6">H6</span>
```

Output: H1 H2 H3 H4 H5 H6

```
<h1>Main Heading 1 <span class="small">Small Heading 1</span></h1>
OR
<h1>Main Heading 1 <small>Small Heading 1</small></h1>
```

Output:

Main Heading 1 Small Heading 1

## Display Headings:

used to stand out more than normal headings  
(larger font-size and lighter font-weight)

```
<h1 class="display-1">Display 1</h1>
<h1 class="display-2">Display 2</h1>
<h1 class="display-3">Display 3</h1>
<h1 class="display-4">Display 4</h1>
```

## More Typography Classes:

.text-start	Indicates left-aligned text
.text-center	Indicates center-aligned text
.text-end	Indicates right-aligned text
.text-justify	Indicates justified text
.text-nowrap	Indicates no wrap text
.text-lowercase	Indicates lowercased text
.text-uppercase	Indicates uppercased text
.text-capitalize	Indicates capitalized text
.font-weight-light	Light weight text
.font-weight-bold	Bold text.
.font-italic	Italic text

## Text Colors:

.text-muted, .text-primary,  
.text-success, .text-info,  
.text-warning, .text-danger,  
.text-secondary, .text-dark,  
.text-light, .text-body,  
.text-light, .text-white

This text is muted.

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

This text represents danger.

Secondary text.

Dark grey text.

Body text.

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

This text represents danger.

Secondary background color.

Dark grey background color.

Light grey background color.

## Bootstrap classes for styling tables:

Class	Purpose
.table	adds basic styling to a table
.table-striped	adds zebra-stripes to a table
.table-bordered	adds borders on all sides of the table and cells
.table-hover	adds a hover effect (grey background color) on table rows
.table-dark	adds a black background to the table
.table-borderless	removes borders from the table

## Contextual classes to color table rows or individual cells:

Class	Description
.table-primary	Blue: Indicates an important action
.table-success	Green: Indicates a successful or positive action
.table-danger	Red: Indicates a dangerous or potentially negative action
.table-info	Light blue: Indicates a neutral informative change or action
.table-warning	Orange: Indicates a warning that might need attention
.table-active	Grey: Applies the hover color to the table row or table cell
.table-secondary	Grey: Indicates a slightly less important action
.table-light	Light grey table or table row background
.table-dark	Dark grey table or table row background

```
<div class="table-responsive">
    <table class="table table-bordered table-hover">
        <!--Html code-->
    </table>
</div>
```

# Bootstrap Images

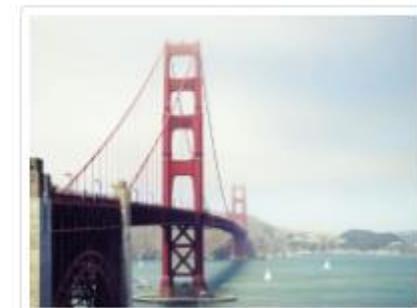
Rounded Corners:



Circle:



Thumbnail:



Class	Purpose
.rounded	adds rounded corners to an image
.rounded-circle	shapes the image to a circle
.img-thumbnail	shapes the image to a thumbnail (bordered)
.float-end	Float an image to the right
.float-start	Float an image to the left
.img-fluid	Create responsive image.(applies max-width: 100%; and height: auto; to the image)
.mx-auto .d-block	Center the image

# Bootstrap alert

Creating an alert with Bootstrap is easy. All we need to do is create a <div> element with classes alert and one of the following contextual state classes.

.alert-success, .alert-info, .alert-warning,  
.alert-danger, .alert-primary, .alert-secondary, .alert-light or .alert-dark

**alert-link** class to any links inside the alert box to create "matching colored links"

**.alert-dismissible** - To close the alert message, add a class to the alert container. Then add **class="close"** and **data-dismiss="alert"** to a link or a button element (when you click on this the alert box will disappear).

```
<div class="alert alert-success alert-dismissible">
  <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
  <strong>Success!</strong> Indicates a successful or positive action.
</div>
```

Click on the "x" symbol to the right to close me.



Badges are used to add additional information to any content.

Ex:

```
<h1>HTML<span class="badge bg-secondary">New</span></h1>
```

## Contextual Badges:

Primary Secondary Success Danger Warning Info Light Dark

## Badge inside an Element

```
<button type="button" class="btn btn-primary">  
    Messages <span class="badge bg-light">4</span>  
</button>
```

Messages 4

## Button Styles:

.btn, .btn-primary, .btn-secondary, .btn-success, .btn-info,  
.btn-warning, .btn-danger, .btn-dark, .btn-light, .btn-link



<b>Button classes can be used on</b>
<a>
<button>
<input type="button">
<input type="submit">

```
<button class="btn">Button</button>
```

```
<p><button class="btn btn-primary">  
btn-primary</button></p>
```

**To make an anchor element look like a button, use `btn` class along with the other button classes (like `btn-default`, `btn-primary` etc.)**

```
<a href="#" class="btn btn-info" role="button">Link Button</a>
```

## Button Outline:

.btn-outline-primary, . btn-outline-secondary, . btn-outline-success, . btn-outline-info, . btn-outline-warning, . btn-outline-danger, . btn-outline-dark, . btn-outline-light



## Block Level Buttons:

```
<div class="d-grid gap-2">  
<button class="btn btn-primary" type="button">Button</button>  
</div>
```

Active/Disabled Buttons: .active, .disabled

Button Sizes: .btn-lg, .btn-sm

# Bootstrap Button Group

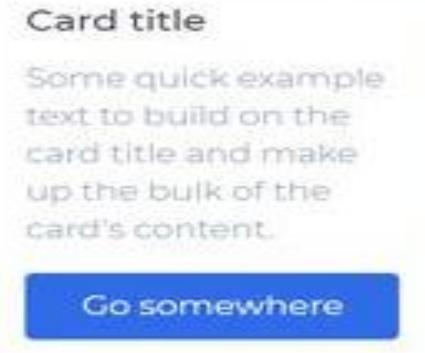
Bootstrap allows you to group a series of buttons together (on a single line) in a button group.

```
<div class="btn-group">
  <button type="button" class="btn btn-primary">Button1</button>
  <button type="button" class="btn btn-primary">Button2</button>
  <button type="button" class="btn btn-primary">Button3</button>
</div>
```

.btn-group-lg|sm can also used to size all the buttons in the group.

.btn-group-vertical to create a vertical button group.

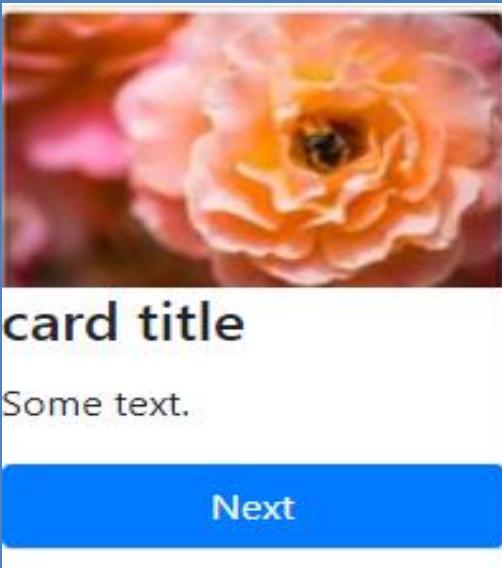
A card in Bootstrap 4 is a bordered box with some padding around its content. It includes options for headers, footers, content, colors, etc.



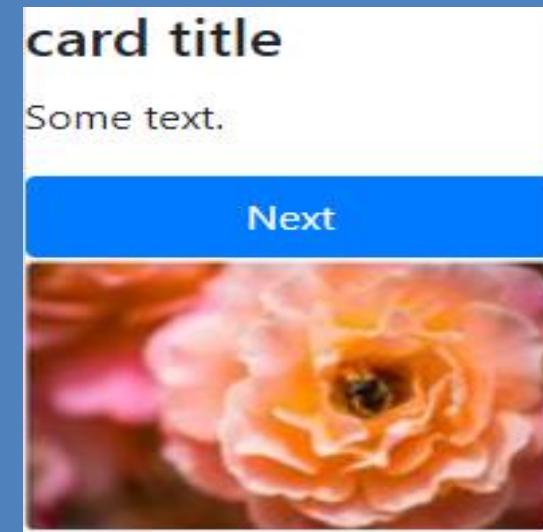
```
<div class="card">  
  <div class="card-header">Header</div>  
  <div class="card-body">Content</div>  
  <div class="card-footer">Footer</div>  
</div>
```

- .card-title: add card titles to any heading element.
- .card-text: used to remove bottom margins for a <p> element if it is the last child (or the only one) inside .card-body.
- .card-link class adds a blue color to any link, and a hover effect.

.card-img-top to an to place the image at the top inside the card.



.card-img-bottom to an to place the image at the bottom inside the card.

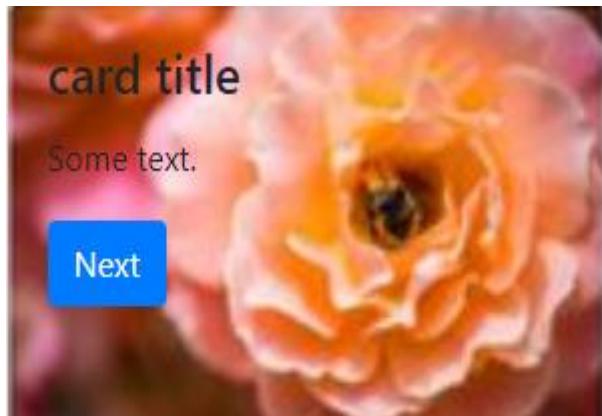


Apply .stretched-link class to a link inside the card, and it will make the whole card clickable and hoverable (the card will act as a link)

```
<a href="#" class="btn btn-primary stretched-link">See Profile</a>
```

## Card Image Overlays:

Turn an image into a card background and use `.card-img-overlay` to add text on top of the image



```
<div class="card" style="width:300px">
  
  <div class="card-img-overlay">
    <h4 class="card-title">John Doe</h4>
    <p class="card-text">Some text.</p>
    <a href="#" class="btn btn-primary">Next</a>
  </div>
</div>
```

`.card-columns` : creates a masonry-like grid of cards (like pinterest).

`.card-deck` class creates a grid of cards that are of **equal height and width**.

`.card-group` class is similar to `.card-deck`. The only difference is that the `.card-group` class removes left and right margins between each card.

# collapse

The **collapse** is used to show and hide another element on the page.

**Ex.** `<button data-bs-toggle="collapse" data-bs-target="#col">Collapsible</button>`  
`<div id="col" class="collapse">`  
This is the example of collapse....  
`</div>`

# Bootstrap Dropdown

For Dropdown we need to include popper.js file:

```
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
```

Class / Attribute	Purpose
dropdown	Specifies a dropdown menu
dropdown-toggle	Specifies the trigger element i.e the element which shows or hides the dropdown menu. In this example the trigger element is the button
data-toggle="dropdown"	This attribute is required on the trigger element to show or hide the dropdown menu
dropdown-menu	The unordered list with this class specifies the dropdown menu items

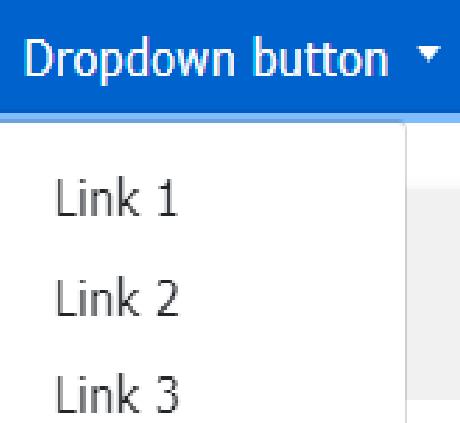
The `.divider` class is used to separate links inside the dropdown menu with a thin horizontal border.

```
<li> <hr class="dropdown-divider"></li>
```

The `.dropdown-header` class is used to add headers inside the dropdown menu.

```
<div class="dropdown-header">Dropdown header 1</div>
```

```
<div class="dropdown">
  <button type="button" class="btn btn-primary dropdown-toggle"
  data-bs-toggle="dropdown">
    Dropdown button
  </button>
  <ul class="dropdown-menu">
    <li> <a class="dropdown-item" href="#">Link 1</a> </li>
    <li> <a class="dropdown-item" href="#">Link 2</a> </li>
    <li> <a class="dropdown-item" href="#">Link 3</a> </li>
  </ul>
</div>
```



## Dropdown Position: .dropright or .dropleft

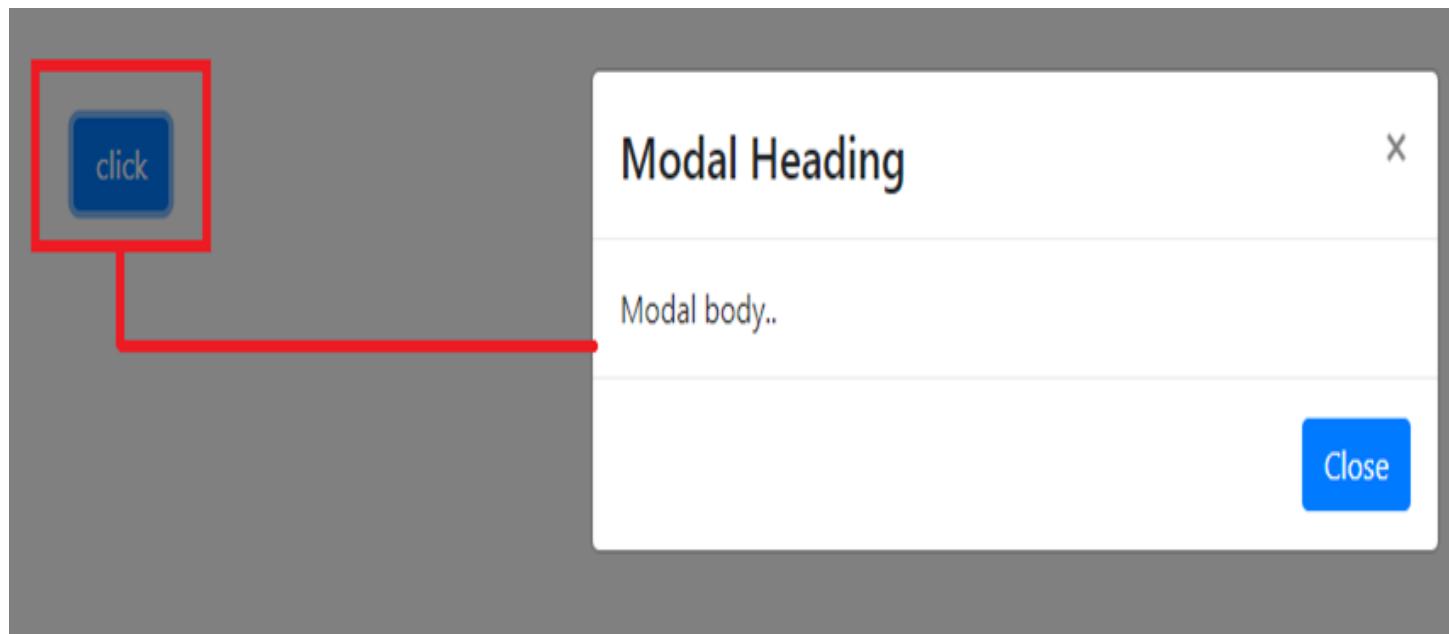
```
<div class="btn-group dropend">
  <div class="btn-group dropstart">
```

**Dropup:** If you want the dropdown menu to expand upwards instead of downwards.

```
<div class="btn-group dropup">
```

# Bootstrap modal popup

The Modal component is a dialog box/popup window that is displayed on top of the current page.



## Attributes and classes to customize the behaviour of the modal:

Class	Description
.fade	Use to add a fading effect when opening and closing the modal
.modal-sm	Creates a small modal.
.modal-lg	Creates a large modal.
.modal-dialog-centered	Center the modal vertically and horizontally within the page

Attribute	Description
data-bs-backdrop="static"	prevents modal closing when clicked outside the modal

```
<button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#myModal">
  Open modal
</button>
<div class="modal" id="myModal">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h4 class="modal-title">Modal Heading</h4>
        <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
      </div>
      <div class="modal-body">
        Modal body..
      </div>
      <div class="modal-footer">
        <button type="button" class="btn-close btn btn-primary" data-bs-
dismiss="modal">Close</button>
      </div>
    </div>
  </div>
</div>
</div>
```

**END**

# Session-7

# Topics to be covered.....

- Bootstrap Forms
- Input Group
- Bootstrap Utility/Helper Classes
- Bootstrap Navs
- Navbar
- Carousel

**Bootstrap provides two types of form layouts:**

- **Stacked (full-width) form**
- **form**

**The following are the classes that are used to style forms:**

Add a wrapper element with **.form-group**, around each form control, to ensure proper margins

**.form-control** - Use this class on all textual elements (<input>, <textarea>, and <select>)

# Bootstrap Stacked Form

Email:

Password:

 Remember meSubmit

```
<form>
  <div class="form-group">
    <label for="email">Email:</label>
    <input type="email" class="form-control">
  </div>
  <div class="form-group">
    <label for="pwd">Password:</label>
    <input type="password" class="form-control">
  </div>
  <div class="form-group form-check">
    <label class="form-check-label">
      <input class="form-check-input" type="checkbox"> Remember me
    </label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

# Horizontal form

Email:

Password:

Remember Me

**Submit**

```
<form>
  <div class="form-group row">
    <label for="email" class="col-form-label col-sm-2">Email:</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="email">
    </div>
  </div>
  <div class="form-group row">
    <label for="pwd" class="col-form-label col-sm-2">Password:</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" id="pwd">
    </div>
  </div>
  <div class="form-group row">
    <div class="col-sm-2 offset-sm-2">
      <div class="form-check">
        <input class="form-check-input" type="checkbox">
        <label class="form-check-label">
          Remember Me
        </label>
      </div>
    </div>
  </div>
  <button type="submit" class="btn btn-primary offset-sm-2">Submit</button>
</form>
```

## Bootstrap Textarea:

```
<div class="form-group">
  <label for="address">Address:</label>
  <textarea class="form-control" rows="5" id="address"></textarea>
</div>
```

## Bootstrap Checkboxes:

```
<div class="form-check"><input class="form-check-input" type="checkbox" value="" id="flexCheckDefault"> <label class="form-check-label" for="flexCheckDefault"> Default checkbox </label> </div>
```

Use the **.form-check-inline** class if you want the checkboxes to appear on the same line

## Bootstrap Radio Buttons:

```
< <div class="form-check"><input class="form-check-input" type="radio" name="flexRadioDefault" id="flexRadioDefault1"><label class="form-check-label" for="flexRadioDefault1"> Default radio </label> </div>
```

```
<input class="form-control" type="file" id="formFile">
```

A standard file input field with a grey border. Inside, the text "Choose File" is in a dark blue font, and "No file chosen" is in a smaller, lighter blue font.

## Bootstrap Select List

```
<select class="form-select" >  
<option selected>Open this select menu</option>  
<option value="1">One</option>  
<option value="2">Two</option>  
<option value="3">Three</option>  
</select>
```

## Switches



Default switch checkbox input

```
<div class="form-check form-switch">
<input class="form-check-input" type="checkbox">
<label class="form-check-label">Default switch checkbox input</label>
</div>
```

## Floating labels:-

```
<div class="form-floating mb-3">
<input type="email" class="form-control" id="floatingInput" placeholder="name@example.com">
<label for="floatingInput">Email address</label>
</div>
```

# Inputs Group

Class	Description
.input-group	A container to enhance an input by adding an icon, text or a button in front or behind the input field as a "help text".
.input-group-prepend	to add the help text in front of the input
.input-group-append	to add it behind the input.
.input-group-text	style the specified help text

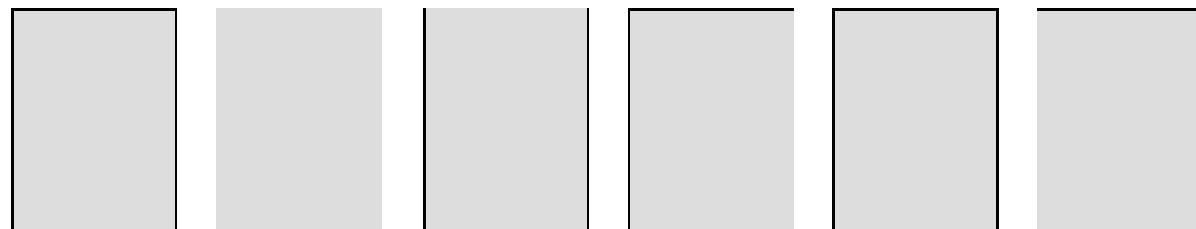
```
<div class="input-group">
    <span class="input-group-text">$</span>
    <input type="text" class="form-control">
</div>
```



Bootstrap 4 has a lot of utility/helper classes to quickly style elements without using any CSS code.

## Borders:

```
<span class="border"></span>
<span class="border border-0"></span>
<span class="border border-top-0"></span>
<span class="border border-right-0"></span>
<span class="border border-bottom-0"></span>
<span class="border border-left-0"></span>
```



## Border color:

```
<span class="border border-primary"></span>
<span class="border border-secondary"></span>
<span class="border border-success"></span>
<span class="border border-danger"></span>
<span class="border border-warning"></span>
<span class="border border-info"></span>
<span class="border border-light"></span>
<span class="border border-dark"></span>
<span class="border border-white"></span>
```



## Border Radius:



.rounded,. rounded-top,. rounded-right,. rounded-bottom,. rounded-left,,  
rounded-circle,. rounded-0

## Width & Height:

.w-25 : Width 25%  
.w-50 : Width 50%  
.w-75 : Width 75%  
.w-100 : Width 100%  
.mw-100 : Max Width 100%

.h-25 : Height 25%  
.h-50 : Height 50%  
.h-75 : Height 75%  
.h-100 : Height 100%  
.mh-100 : Max Height 100%

# Spacing:

Bootstrap 4 has a wide range of responsive margin and padding utility classes.

The classes are named using the format :

{property}{sides}-{size} for `xs` and

{property}{sides}-{breakpoint}-{size} for `sm`, `md`, `lg`, and `xl`.

Where *property* is one of:

`m` - for classes that set `margin`

`p` - for classes that set `padding`

Where *sides* is one of:

- t** - for classes that set `margin-top` or `padding-top`
- b** - for classes that set `margin-bottom` or `padding-bottom`
- s** - for classes that set `margin-left` or `padding-left`
- e** - for classes that set `margin-right` or `padding-right`
- x** - for classes that set both `*-left` and `*-right`
- y** - for classes that set both `*-top` and `*-bottom`
- blank - for classes that set a `margin` or `padding` on all 4 sides of the element

Where *size* is one of:

- 0** - for classes that eliminate the `margin` or `padding` by setting it to 0
- 1** - (by default) for classes that set the `margin` or `padding` to `$spacer * .25`
- 2** - (by default) for classes that set the `margin` or `padding` to `$spacer * .5`
- 3** - (by default) for classes that set the `margin` or `padding` to `$spacer`
- 4** - (by default) for classes that set the `margin` or `padding` to `$spacer * 1.5`
- 5** - (by default) for classes that set the `margin` or `padding` to `$spacer * 3`
- auto** - for classes that set the `margin` to `auto`

## Display property:

To make an element into a block element.

As such, the classes are named using the format:

- .d-{value} for xs
- .d-{breakpoint}-{value} for sm, md, lg, and xl.

```
<span class="d-block bg-success">d-block</span>
<span class="d-sm-block bg-success">d-sm-block</span>
```

```
<div class="d-inline p-2 bg-primary text-white">d-inline</div>
<div class="d-inline p-2 bg-dark text-white">d-inline</div>
```

d-inline

d-inline

## Nav Menus:

```
<ul class="nav">
  <li class="nav-item">
    <a class="nav-link" href="#">Home</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">About</a>
  </li>
</ul>
```

Home      About Us

## Aligned Nav:

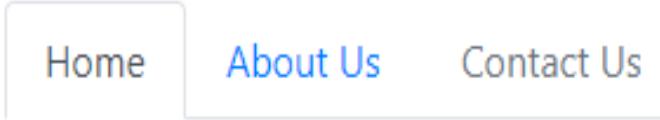
- .justify-content-center class to center the nav.
- .justify-content-end class to right-align the nav.

## Vertical Nav: <ul class="nav flex-column">

Home

About Us

### Tabs:



```
<ul class="nav nav-tabs">
  <li class="nav-item">
    <a class="nav-link"
active" href="#">Home</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">About
Us</a>
  </li>
  <li class="nav-item">
    <a class="nav-link"
disabled" href="#">Contact Us</a>
  </li>
</ul>
```

**Pills:** <ul class="nav nav-pills">

Home

About Us

Contact Us

.nav-justified: Justify the tabs/pills.

**Fill and justify:**

<ul class="nav nav-pills nav-fill">

# Bootstrap navbar

A navigation bar is a navigation header that is placed at the top of the page.

```
<nav class="navbar navbar-expand-sm bg-secondary navbar-light">

  <ul class="navbar-nav">
    <li class="nav-item active">
      <a class="nav-link" href="#">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">About</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Contact</a>
    </li>
  </ul>

</nav>
```

Home About Contact

## Vertical Navbar:

Remove the `.navbar-expand-xl|lg|md|sm` class to create a vertical navigation bar.

Class	Description
<code>.justify-content-center</code>	center the navigation bar
<code>.navbar-brand</code>	used to highlight the brand/logo/project name of your page
<code>.fixed-top</code>	makes the navigation bar fixed at the <b>top</b>
<code>.fixed-bottom</code>	make the navbar stay at the <b>bottom</b> of the page
<code>.sticky-top</code>	make the navbar fixed/stay at the <b>top</b> of the page when you scroll <b>past</b> it
<code>.navbar-dark</code>	Add a <b>white</b> text color to all links in the navbar
<code>.navbar-light</code>	add a <b>black</b> text color to all links in the navbar
<code>.navbar-text</code>	Use to vertical align any elements inside the navbar <b>that are not links</b>

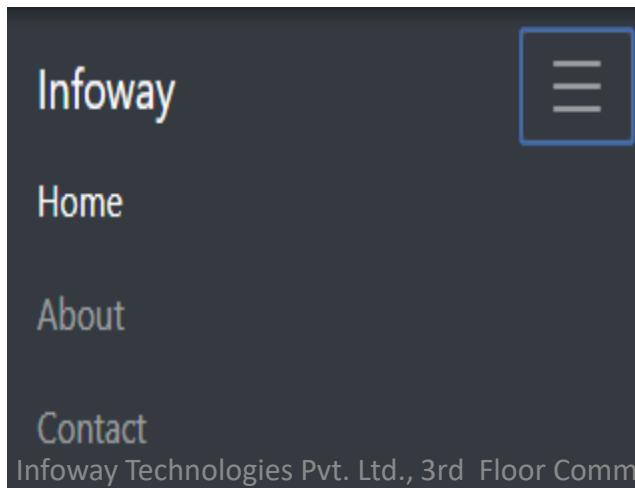
# Bootstrap navbar component

It is useful for creating responsive navigation header for a website. When the toggle button is clicked the navbar expands. However, on large screen devices like laptops and desktops, where we have enough room to display the entire navbar, it appears normal.

**Creating responsive navbar :** The navbar appears normal on a large screen size devices like laptops, desktops etc.

Infoway Home About Contact

The navbar appears collapsed on a small screen size devices like mobile phones, portrait tablets etc. The navbar expands when the toggle button is clicked.



```
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
    <!-- Brand -->
    <a class="navbar-brand" href="#">Infoway</a>

    <!-- Toggler/collapse Button -->
    <button class="navbar-toggler" type="button" data-bs-
        toggle="collapse" data-bs-target="#colNav">
        <span class="navbar-toggler-icon"></span>
    </button>

    <!-- Navbar links -->
    <div class="collapse navbar-collapse" id="colNav">
        <ul class="navbar-nav">
            <li class="nav-item active">
                <a class="nav-link" href="#">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">About</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Contact</a>
            </li>
        </ul>
    </div>
</nav>
```

# Bootstrap Carousel Plugin

The Carousel is a slideshow for cycling through elements.



```
<div id="myCarousel" class="carousel slide"  
data-bs-ride="carousel">
```

Class	Description
.carousel	Creates a carousel
.carousel-indicators	Adds indicators for the carousel. These are the little dots at the bottom of each slide (which indicates how many slides there are in the carousel, and which slide the user are currently viewing)
.carousel-inner	Adds slides to the carousel
.carousel-item	Specifies the content of each slide
.carousel-control-prev	Adds a left (previous) button to the carousel, which allows the user to go back between the slides
.carousel-control-next	Adds a right (next) button to the carousel, which allows the user to go forward between the slides
.carousel-control-prev-icon	Used together with .carousel-control-prev to create a "previous" button
.carousel-control-next-icon	Used together with .carousel-control-next to create a "next" button
.slide	Adds a CSS transition and animation effect when sliding from one item to the next. Remove this class if you do not want this effect

**END**

# Thank You