

FRONTEND

HTML [HyperText Markup Language]

What is a Web?

- Internet is a wide area network that connect computers all over the world.
- In early 1990's Tim Berners Lee introduced the concept of "Web".
- Web is a portion of Internet with Restricted access.
- W3C [World Wide Web Consortium] maintains standards of web.
- The latest version of Web is Web-3.

What is HTML?

- HTML stands for HyperText Markup Language
- Hypertext → It can you take the location beyond you see.
- Hypertext refers to the text that takes user beyond the context when what he view on screen.
- Markup is the term derived for "markup", which refers to presentation.
- Markup Language is a language for presentation.
- HTML is a presentation language or not a programming language.

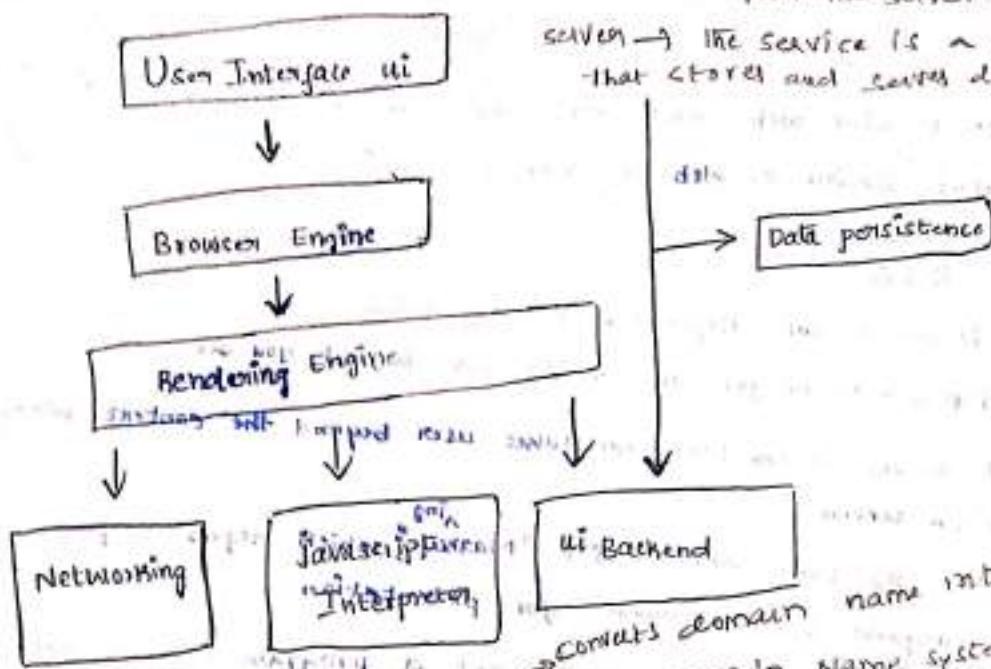
Evolution of HTML

- CERN labs developed a language called GML (Generic Markup Language) for internet.
- CERN improved GML with more features and introduced as SGML [Standard Generic Markup Language]
- In early 1990 Tim Berners Lee introduced HTML
- IETF [Internet Engineering Task force] developed HTML upto version 3.1
- In early 2004 WHATWG took responsibility of HTML and started evolving HTML (Web HyperText Application Technology Work Group)
- WHATWG started with HTML 4 and latest version is HTML 5 (2014)
- Latest version of HTML is HTML 5.

What is Web Browser (or) Browser?

- It is a software tool used by users to view Hyper Text documents that present content on web.
- Internet → Internet is a global network that connects millions of computers which allows (text, images, video) to be shared via services.
- Ex: chrome, Firefox, safari, edge etc.

Browser Architecture



Client → Client means browser (chrome etc)

- It requests data (or) service from the server.

Server → The service is a computer that stores and serves data.

data

Data persistence

Converts domain name into IP address

DNS → Domain Name System

(↳ Human-readable address of website)

(157.240.22.32.5)

www.facebook.com ✓

IP2. In structured form DNS → www.157.240.22.32.5

[Expands gradually]

1) User Interface (UI) :

It provides the UI for browser, which includes titlebar, menubar, short cut button, address bar, extensions, favicons etc.

2) UI Backend :

It stores the data required for browser, which including user credentials, Recently viewed documents etc.

3) Browser Engine :

It translates HTML and CSS.

4) Rendering Engine :

It is responsible for Generating Output

5) Javascript Interpreter :

It is responsible for translating javascript line by line

6) Network :

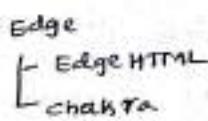
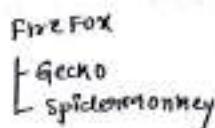
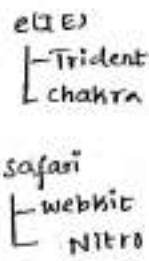
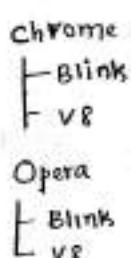
It is responsible for tracking the performance of page

7) Data Persistence :

It stores data of various websites you access in browser. It includes cookies, local storage, session etc.

Browser Engines:

- v8
- webkit
- gecko
- spider monkey
- chakra
- nitro etc



Website :

It is a virtual directory on web server that provides access to various resources

Every website comprises of 2 paths

a) physical path [D:\flipkart]

b) virtual path http://127.0.0.1:5500 (www.name.com)

All resources are kept in physical path

Resources are accessed by using virtual path

Webpage :

It is a hypertext document, that provides an UI from where user can interact within the source of website

Tokenization

The process of converting chars into tags

- HTML is used to create static and dynamic webpages
- HTML is a collection of elements arranged in an hierarchical order called DOM [Document Object Model]

- HTML parsing

- Markup \Rightarrow Bytes \Rightarrow chars \Rightarrow Tokens \Rightarrow Elements \Rightarrow DOM \Rightarrow Layout \Rightarrow Render \Rightarrow paint

NOTE:

In browser developer tools you can view the perform

a) performance

b) Network

HTML is a set of elements arranged in hierarchy called "DOM"

HTML elements are classified into 5 groups.

1) Normal Elements

2) void Elements

3) the Data Elements

4) RawText Elements

5) Foreign Elements

1) Normal Elements

A Normal Element Returns presentation directly on callback.

Normal Element can present the presentation without using any additional attributes.

It starts presentation but will not stop implicitly. You have to manually stop by using an end token

$<>$ start token

$</>$ end token

$$ start

$$ end

2) Void Elements

- The term void refers to no return type.
 - Void Element cannot Return
 - It Required additional attributes.
 - Void elements are self ending elements, hence does not require end tokens.
- Ex:

3) Rc Data Elements

- Rc Data Elements → Rich Content Data Elements.
- These elements will not allow any other element with in the content.
- These are used to present text content

Ex: <textarea> </textarea>

4) Raw Text Elements

- These are the elements present using raw text
- A token is not required for these Elements [Tags are not Required]

Ex: 8#8377;

copy;

5) Foreign Elements

- These are the elements, which are not native to browser.
- They need additional library
- You can import Relative library and use the Elements

Ex:

SVG
Canvas
mathML

Static & Dynamic Web pages

- static Refers to continuous memory.
- The memory allocated for first object will Continue for other objects.
- A static page Comprises of same information to send across multiple Requests.
- static page contains Extensions :-
 - .html
 - .htm

- Dynamic Refers to the discrete memory.
- Dynamic Page is also called as Non-static Page.
- Memory is newly allocated for every Object.
- Memory is newly allocated for every client request.
- a dynamic page have customized Response for every client request.
- Dynamic page contains extensions :-
 - .asp
 - .aspx
 - .jsp
 - .php etc

Note:
HTML is a language used for designing both static and dynamic pages.

Designing a static page

- i) Every static page must have extension
 - .html
 - .htm

FAQ: What is difference between html & htm?
Both are same

- ii) Every static page starts with Document Definition

<!DOCTYPE html /> → It ~~tells~~ tells to the Version of the html to the browser

{
 {<> Token Tag
 {<!> Not Tag
 }

3) Every static page comprises of document scope

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

4) Every document scope must mention the language type for content in page

" lang = en-in | en-uk "

Syntax :

```
<!DOCTYPE html>
<html lang = "en-in">
</html> \ attribute
```

5) Every HTML document scope Comprises of 2 Sections

(a) <head>

b) <body>

Head section - It defines content which is intended to load into browser memory and later deliver to page.

Typically every HTML page head section comprises of

- a) Title → It will display title for page in title bar
- b) Link * It is used for bookmarking page
- c) meta * It used for Search Engine Optimisation (SEO)
- d) script
- e) style

a) Title

- Title in title bar
- Bookmarking
- SEO

b) Link

- It is used to link any external document to your page.
- Usually it can be a shortcut icon or stylesheet

→ creating a shortcut icon for website

i) Folder "public/Images"

ii) Add new file into Images folder by name "favicon.ico".

3) Open mspaint in windows

4) Open your favicon in mspaint

file = Open \Rightarrow d:\flipkart\public\images\favicon.ico

5) Goto file menu \Rightarrow image properties \Rightarrow set size 32x32 pixels

6) draw your icon and save [control + s]

7) Go to your project index.html page

```
<head>
<link rel = "shortcut icon" href = "public/images/favicon.ico">
</head>
```

3) Meta

* Meta Refers to "Meta Data".

* It contains information about your page to browser and SEO.

* Meta is used to make to page

a) responsive

b) SEO Friendly

Meta Responsive Attributes:

i. view port : It identifies the dimensions of client device and fits the content of page according to device.

```
<head>
<meta name = "viewport" content = "width = device-width, initial-scale = 1">
</head>
```

viewport : Refer to device and its screen size

initial-scale : Refer to zoom % of page [$1 = 100$]

j. http-equiv

- It defines how a page need to be requested over http [Hypertext Transfer Protocol]

- If `http-equiv = refresh` is used to reload a page at regular time intervals

Syntax :

```
<head>
<meta http-equiv = "refresh" content = "5" >      5 = 5 sec
</head>
```

- NOTE: If only specific area in page is reloading, then it's called AJAX.
- b) Meta SEO Friendly Attributes:
- SEO stands for Search Engine Optimization
 - + Web crawlers and Web spiders are responsible for identifying the search pattern of users and return the results from database. [Matching the pattern website]
1. keywords
- <meta name = "keywords" content = "some keywords that are used to find our page">
2. Description
- <meta name = "description" content = "some summary about your website">
- 3) Charset
- + It defines the language type used for Webpages
 - + It specifies how much space is needed for processing the page according to characters set.
 - + Charset is defined according to UTF standards
[UNICODE TRANSFORMATION FORMAT] UTP

UTF-8	8 Bits	- English
UTF-16	16 Bits	- Korean, Japanese
UTF-32	32 Bits	- Chinese
UTF-64	64 Bits	- Arabic
- Syntax:
- <meta charset = "utf-8">
- 4) style
- It is used to embed style attributes in a web page
- Syntax:
- <style type = "text/css">
 </style>
- 5) script
- It is used to embed client or server side script into web page
- Syntax:
- <script type = "text/javascript">
 </script>

Body section:

* It defines the content which is directly rendered into page

Syntax:

```
<body>
  .... Your Content ...
</body>
```

Attributes:

1) bgcolor : It sets background colour for page

2) text : It sets color for text in page.

(Q) How to define colours in HTML?

- * (a) Color Name } In HTML colors can be defined in 3 ways.
 (b) Color shade Name }
 (c) Hexa Decimal Code

```
<body bgcolor = "yellow">
<body bgcolor = "lightyellow">
```

(ii) Hexa decimal code:

1. 3 char code with #

2. 6 char code with #

[Red, Green, Blue]

#RGB

#RRGGBB

R, G, B

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f

0 => Dark
f => Bright

#0f0 Green

#00ff00 Green

#000000 Black

#ffff White

#ffff White

Syntax:

```
<body bgcolor = "#ff00" text = "#ffff"for Background      for text
```

Body section Attributes

1. bgcolor
2. text
3. background
4. align
5. alink
6. vlink
7. leftMargin
8. RightMargin
9. BottomMargin
10. TopMargin

3) background : It sets a background image for page.

Note: Background will overwrite the background color

Syntax:

```
<body background = "public/images/girl.jpg">
</body>
```

Note: HTML can set background image, but cannot control the image layout
we have to use CSS background attributes.

background-size	— width and height in pixels
background-attachment	— fixed, scroll
background-position	— top, center, left, right, bottom
background-repeat	— no-repeat, repeat-x, repeat-y

Syntax:

```
<head>
<style>
```

body {

```
background-size: cover;
background-repeat: no-repeat;
background-position: center center;
background-attachment: fixed
```

}

```
</style>
</head>
```

```
<body background = "public/images/girl.jpg">
</body>
```

4) a:link : It sets color for active links in page

5) a:visited : It sets colour for visited links in page

Syntax:

```
<body a:link = "red" a:visited = "green">
<a href = "http://www.amazon.in"> Amazon </a>
</body>
```

6) align : It aligns the content left, center, right (or) justify

Syntax:

```
<body align = "center | justify | left | right">
</body>
```

- 7) LeftMargin
 - 8) rightMargin
 - 9) topMargin
 - 10) bottomMargin
- } Sets space b/w content and browser

If size of web page is 1200px

Note:

Every webpage max-width is 1200px

min-width for content must be 500px

Syntax:

```
<body leftmargin="350" rightmargin="350" topMargin="50" bottomMargin="50">
</body>
```

Body SEMANTICS:

- * Semantic refers to element meant for a specific purpose
- * The name defines what its purpose is.
- * HTML have over 100 semantics
- * Body section semantics used for designing a layout are:

1. <header>
2. <footer>
3. <section>
4. <main>
5. <narr>
6. <articles>
7. <aside>
8. <aside>
9. <dialog>
10. <figure>
11. <figcaption>
12. <div>
13.

} HTML 5 semantics

Note: HTML5 introduced the new semantics to make the page more SEO and user friendly.

SEMANTIC	Description
1. <header>	: Content to display to at top margin of page
2. <footer>	: Content to display at bottom margin of page
3. <section>	: It defines the content display with header and footer
4. <main>	: It defines content used as entry point
5. <article>	: It defines highlights of your page
6. <aside>	: It contains the information that is not relative to current page
7. <nav>	: It defines navigation area
8. <dialog>	: It configures a pop-up
9. <figure>	: It encapsulates any image (or) graphic with caption
10. <figcaption>	: It sets SEO friendly caption for image.
11. <div>	: It is a division
12. 	: It spans the content along with existing content, however it can separate the content to highlight.

Q) Why to use so many semantics instead of one container?

Ans To Configure "Accessibility".

Bootstrap Icons for Web Design:

1. Open your project in VSCode
2. Open Terminal and switch it into cmd prompt
3. Run the following command
`npm install bootstrap-icons --save`

4) This will add a new folder into your project by name "node-modules".
 5) All icons are present in a CSS file
`node-modules/bootstrap-icons/font/bootstrap-icons.css`

- 6) Link the CSS file to your page

```

<head>
  <link rel="stylesheet" href="../../node-modules/bootstrap-icons/font/bootstrap-
  icons.css">
</head>
  
```

- iii) Apply text using `<div>` & `` with bootstrap icon class name
- open class = "list-group-item" in `<div>`
 - class = "list-group-item" in ``

i) Header:

Header consists of brand name, brand logo, shortcut buttons, quick search etc.

CSS Text styles:

```
.font-weight : bold;
.font-style : italic;
.font-align : left, center, right, justify;
.fixed-size : in pixels;
color : text-color;
.font-family : Arial;
```

CSS Display flex:

`display` : It keeps the element flexible in container
 you can arrange in Row or Column
 It allows the to justify the container.
`justify-content` : It keep space b/w, space around, space evenly, centre,
`(flex-start, flex-end)`

ii) Nav

It is used for configuring navigation area in image

Item in nav can be defined using ``

Syntax:

```
<nav>
<span> Home </span>
<span> About </span>
<span> Contact Us </span>
</nav>
```

Article

- It defines the highlights of page
 you can keep latest trending news of your page in article

Syntax:

```
<article>
  Offers ... highlights ...
</article>
```

~~as~~ aside:

- It defines the content which is not relative to current context.
- It navigate the user to partner business

Ex: Advertisements

Syntax:

```
<aside>
  ... content to navigate outside current context
</aside>
```

CSS Absolute and Fixed positions:

- Position: absolute → It removes element from its position and keeps with relative to content in page. It will scroll along with content.
- position: fixed → It removes elements from its position and keeps relative to browser. It is locked with scroll.
- top, left, right, bottom → These are used for define the position of the element in page

Syntax:

aside
If you want scroll the Icons then you will use it.

```
{  
  position: absolute | fixed;  
  bottom: 10px;  
  right: 10px;  
}
```

4) Section:

- It defines the content to display b/w header and footer
- the page content is usually kept in section

3) Main

* It Refers to entry point

* In a website "main" defines from where user will start using the site

Syntax:

```
<header>
</header>
<Section>
<main>
</main>
</Section>
<footer>
</footer>
```

Note:

To set transparent colors you can use CSS method "rgba()

Syntax:

rgba (red, green, blue, alpha)

red, green, blue \Rightarrow 0 to 255

alpha \Rightarrow 0 to 1

Ex: rgba (0, 255, 0, 0.6) \Rightarrow Green Transparent color

FAQ: How to set fluid width and height? It changes according

to screen width & height

Ans: By defining in "%", "vh", or "vw".

vh \Rightarrow view port height $100 = 100\%$

vw \Rightarrow view port width $100 = 100\%$

```
{  
width : 100vw;  
height : 100vh;
```

}

2) Dialog

- It is used to design pop-up content in page.
- It is used hidden in state and you can open when required by using "open" attribute

Syntax:

```
<dialog open>
  .... your content ...
</dialog>
```

3) Figure

- It is used to display the images and graphics with caption
- "figurecaption" makes the image SEO friendly.

Syntax:

```
<figure>
  ... your image ...
  <figurecaption> Caption </figurecaption>
</figure>
```

HTML body Semantics and Entities

1) Line Break:

It is defined by using `
`

FAQ: What is difference between `
` & `</br>`?

Ans: There is no `</br>` in HTML. It is just developer's tradition to define like

Self ending elements [void Elements].

Functionality `
` and `</br>` will be same.

2) Blank Spacing

- Blank space means (white space) or simply means the empty space b/w words, elements, characters.

- They are defined by using " " [non-breakable space].

Syntax:

Words Words.

(3) Formatted Text:

- It keeps the formats in page exactly as defined in editor.
- Formatted Text means the text will appear in the browser exactly the way you typed in your HTML file - with all spaces, tabs, line breaks.
- It is defined by using `<pre>` element

Syntax:

```
<pre>
line-1
line-2
line-3
line-n
</pre>
```

(4) Variables:

- `<var>` tag is used to represent the variable name in mathematical expressions, programming code, or formulas.

Note:

It does not store any value (like in Java), it just gives semantic meaning.

Ex Text:

Syntax:

```
<var>xyz </var>=10;
```

5) Sample Text:

- It is used to represent output from a computer programme, like console output or example values.
- It is defined by using `<samp>` element

Syntax:

```
<samp> Write a programme for palindrome </samp>
```

6) Code Block:

- It is used for to display the programming code in Web page.
- It is defined by using `<code>` element

Note:

`<code>` → for inline (single line)

`<pre> + <code>` → for multiple line

Syntax:

```
<code>
class demo
{
}
</code>
```

7) Large Text:

Large Text refers to text that is displayed bigger than normal Text

Note:

- In old HTML, `<big>` was used, but it is now deprecated.
- In modern HTML5, Large Text is created using headings tags (`<h1>` to `<h6>`)

Syntax:

```
<big> Title </big>
```

8) Small Text:

- Small text refers to text is displayed smaller than normal Text.
- `<small>` tag is used to make text smaller than the surrounding Text.

↓
deprecated

Syntax:

```
<small> Subtitle </small>
```

9) Details and summary

- It is used to create expandable and collapsible content.

`<details>` → defines a section that the user can open or close

`<summary>` → defines the heading (title) for the `<details>` section, which is always visible

When clicked, `<details>` expands to show hidden content

Syntax:

```
<details>
<summary> Title </summary>
...
...
...
</details>
```

HEADINGS IN HTML

- Headings in html are used to define titles and subtitles on a web page
- they are represented by `<h1>` to `<h6>`
- headings are defined by using `<hn>` element, where "n" refers to level number
 - 1 to 6
- `<h1>` - Main heading (largest, most important)
- `<h2>` - subheading under `h1`
- `<h3>` - sub-subheading under `h2`
- `<h4>` - small heading under `h3`
- `<h5>` - even smaller
- `<h6>` - smallest heading (least important)

Syntax:

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

FAQ:
We can format any text as heading without using heading style, then why we to use heading element?
So it make headings SEO friendly, so that web spiders can make summary of topics in page.

FAQ:
Can we modify the heading text by using style?

Ans: Yes. It includes font-style, size etc.

FAQ: Can we remove any specific style defined for heading?

Ans: Yes. By using CSS "UNSET" value.

Syntax: Specific

Syntax: All

```
h1
{
    font-weight: unset;
    font-size: unset;
    display: unset;
}
```

```
h1
{
    all: unset;
}
```

Note:

- Don't use too many headings in web page. It may SPAM (Unwanted messages)
- Don't use headings for highlighting an word or sentence in a paragraph, it may cause of violation SEO rules.

Syntax:

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

FAQ:

We can format any text as heading without using heading style, then why we to use heading element?
 So to make headings SEO friendly, so that web spiders can make summary of topics in page.

FAQ:

Can we modify the heading text by using style?

So? Yes. It includes font-style, size etc.

FAQ) Can we remove any specific style defined for heading?

So? Yes, by using CSS "UNSET" value.

Syntax: Specific

```
h1
{
  font-weight: unset;
  font-size: unset;
  display: unset;
}
```

Syntax: All

```
h1
{
  all:unset;
}
```

Note:

- 1) Don't use too many headings in web page. It may SPAM (unwanted messages)
- 2) Don't use headings for highlighting a word or sentence in a paragraph, it may cause of violation SEO rules.

Paragraphs and Blockquotes:

- Defines a block of text (a paragraph)
- Browser automatically add some space before and after a `<p>`
- The paragraphs are defined by using `<p>` element.
- (Blockquotes)-
- ↳ Used to quote text from another source
- ↳ by default, browser display blockquotes indented (moved inside)
- by default, browser display blockquotes "indented (moved inside)"
- Blockquotes is similar to paragraph but have left and Right padding.
- Blockquotes used to summarise the content in the page. `<blockquote>`
- Blockquotes used to summarise the content in the page.
- Paragraphs and Blockquotes designate a container, which can be aligned in page.

Syntax:

```

<p align = "center"> some text </p>
<blockquote align = "center">
    ...
    text
    ...
</blockquote>

```

Data List with Terms and Definitions

~~~~~

- A data list defined by using `<dl>` element
- List contains terms defined with `<dt>` element
- Every term comprises of definitions of defined with `<dd>` element
- `<dl>` — definition list (container)
- `<dt>` — definition Term (the word/concept)
- `<dd>` — definition Description (the explanation / definition of the term)

In short:

`<dl>` = list, `<dt>` = word, `<dd>` = meaning

Syntax:

```
<dl>
  <dt> Term-1 </dt>
  <dd> Definition </dd>
  <dt> Term-2 </dt>
  <dd> Definition </dd>
</dl>
```

FAQ:

1) How to set DropCap for paragraph?

By using CSS class " ::first-letter" and CSS attributes like "size and float".

Syntax:

```
p::first-letter
{
  font-size: 55px;
  font-family: alegreya;
  float: left;
}
```

2) How to set space between lines words and chars in paragraph?

By using following CSS attributes

line-height.

word-spacing

letter Spacing.

3) How to design sticky data terms

By using CSS "Position: sticky"

Syntax:

```
dt
{
  position: sticky;
  top: 0px;
}
```

4) How to set firstline "indent" for paragraph?

By using CSS attribute "text-indent"

Syntax:

```
p f
  text-indent: 200px;
}
```

9/9/2025

**LISTS :**

- An HTML List allows you to organize that data on Web page into an Ordered or unordered format to make the information easier to read and visually appealing.
- HTML lists are very useful for creating structured, accessible content in web dev.
- `<li>` tag is used to define each item of the list & syntax:  
`<li> Content </li>`

**Types of HTML Lists:**

- Ordered List (`<ol>`): Items in a list that are listed in Order of numbers or letters

2) Unordered List (`<ul>`): Bullet items which do not have a natural ordering

- Unordered List (`<ul>`): Bullet items which do not have a natural ordering

- Description List (`<dl>`): Term-description association such as glossary or dictionary

**1) Ordered List:**

The HTML Ordered List (or Numbered List) is applied to present items list items in a order.

It sets auto numbering for list items

- It sets auto numbering for list items
- the ordering is given by a numbering scheme, Arabic numbers, Letters, Roman numbers.

List is defined by `<ol>`

Items are defined by `<li>`

**Syntax:**

`<ol>`

`<li> item-1 </li>`

`<li> item-2 </li>`

`<li> item-3 </li>`

`</ol>`

- \* We can also change the numbering style by using "type" attribute.

| Type | Description                             |
|------|-----------------------------------------|
| "1"  | Default, Numbers (1,2,3...)             |
| "A"  | Uppercase Letters (A,B,C,D...)          |
| "a"  | Lowercase Letters (a,b,c,d...)          |
| "I"  | Uppercase Roman numerals (I,II,III,...) |
| "i"  | Lowercase Roman numbers (i,ii,iii...)   |

Syntax:

<ol type="A">

<ol type="a">

- \* We can change numbering define from where numbering to start using "start"

attribute

Syntax

<ol type="1" start="5"> \* start's a level number

- \* We can reverse the numbering by using "reversed" attribute.

Syntax:

<ol type="1" reversed>

- \* We can create nested Ordered Lists. every nested list must be the inside the <li> element.

Syntax:

<ol>

<li> parent </li>

<ol type="a">

<li> child </li>

<li> child </li>

</ol>

</li>

</ol>

FAB)

i) how many arrange list items side by side  
By using `display: flex` or `display: inline`

`flex`: It is used for `<ol>` with numbering

`inline`: It is used for `<li>` without numbering.

Syntax:

- options li {

`display: inline`

    }      how to keep list items in multiple columns ?

ii) how to keep list items in multiple columns ?  
By using & styles attributes

a) `display: grid`      sets numbering      order left to right

order top to bottom

b) `columns`:      sets numbering

Grid :      columns:  
 1      2      1      3  
 3      4      2      4

iii) how to create a scrollable list ?

By using following CSS attributes

- border
- height
- overflow

Syntax:  
`ol {`  
`border: 1px solid gray;`  
`width: 60px;`  
`overflow: auto;`  
`}`

iv) how to generate numbering for ordered list ?

By using "list-style" attribute set to ""none"".

Syntax:

```
ol
{
    list-style: none;
}
```

## UnOrdered List

- An unOrdered list defines a list of items in which the Order of the items does not matter.
- It is also called as Bulleted list.
- These are very popular in webpages where they are used in displaying menus, feature, sets, grouped information etc.
- An Unordered list tag is used to create an unordered list.
- Items are defined by `<li>` tag.
- An unordored list defined by `<ul>` tag.
- List type can be "circle", "square", "disc".

Syntax:

```

<ul>
  <li> item-1 </li>
  <li> item-2 </li>
</ul>

```

-there are 4 types of style in unorderd lists:

- type = "disc" - sets the list item marker a bullet (default).
- type = "circle" - sets the list item marker to a circle.
- type = "square" - sets the list item marker to a square.
- type = "none" - the list items will not be marked.

Syntax:

```

<ul type="square">
  <li> item-1 </li>
  <li> item-2 </li>
</ul>

```

To specify the bullet style will be used as the list item marker.

\* We can set a custom symbol as bullet symbol, you have to use "list-style-image"

attributes of CSS.

Syntax:

- Create a new image with 20x20 px in my paint.
- draw and save your icon as a GIF type.
- Set for `<ul>`

415

```
list-style-image: url("images/star.gif");  
}
```

Description List:

- Description List :  
• Description list is a list in which each term contains its description.

This contains `<ads>` tags.

- 2)  $\langle dd \rangle \langle /dd \rangle \rightarrow$  It is used to describe the term  
 3)  $\langle dl \rangle \langle /dl \rangle \rightarrow$  It is used to define the term

## Text Formating in HTML :

- \* HTML Text Formating refers to the modify the appearance and structure of text on a webpage.

text on a webpage.

- HTML provides us ability to format ...
  - Text Formatting allows you to style text in different ways, such as the italic, underlined, highlighted.

**physical tag:** these tags are used to provide the visual appearance to text  
and are used to add some logical or semantic value to

**physical Tag**: these tags are used to add some logical or semantic value to  
**Logical Tag**: these tags are used to add some logical or semantic value to

-the text description

Element name

- Element name  
1) <bs> → this is a physical tag, which is used to hold the text in written between it

- ② <b> → this is a logical tag, which tells the browser that the text is important between it
- ③ <strong> → this is a logical tag, which tells the browser that the text is important and to make text as italic

2) <strong> → It is a physical tag which is used to make text as bold.

3) <i> → this is a physical logical tag "display content Italic."

40 <em> n n n logical tag </em>

5) `links` → It is used to display the content which is added.

6) `edel` → It is used to delete display the deleted content

- 7) `<sup>` It displays the contents slightly above the normal line.
- 8) `<sub>` It displays the contents below the normal line.
- 9) `<big>` Increases the font size.
- 10) `<small>` Decreases the font size.
- 11) `<u>` It is used to underline text written between it.
- 12) `<mark>` It is used to highlight text.
- 13) `<strike>` It is used to display the content with strikeout.

### Horizontal line:

- `<hr>` tag is used to create a horizontal line.
- It is a self-closing tag and does not require an end tag.

### Syntax:

`<hr size="3" color="blue" width="500" nospace>` → specify the bar without shadow effect.

### Images in html

- \* Images are significant in improving the attractiveness and interactivity of website.
- \* `<img>` tag applied to insert images in a web page and it does not end tag.

### Syntax:

``

Web supports only following various types of image.

1) APNG

image/apng

Animated portable Network Graphics [·apng] ..

- HD image

Portable Network Graphics [·png]

2) PNG

image/png

Joint photographic Expert Group [·jpeg]

3) JPEG

image/jpeg

Graphic Interchange format [·gif]

4) GIF

image/gif

Tagged image File format [·tiff]

5) TIFF

image/tiff

|         |            |                               |
|---------|------------|-------------------------------|
| 6) BMP  | image/bmp  | Bitmap [-bmp]                 |
| 7) ICO  | image/ico  | microsoft Icon [-ico]         |
| 8) SVG  | image/svg  | scalar vector graphics [-svg] |
| 9) WebP | image/webp | web picture [-webp]           |

FAQ):

When to use specific image type? what are the its features & applications?

APNG & PNG - HD images, more pixel depth, but it occupies more memory.

- If you are defining images for download then prefer on png.

JPG - HD image, compressed images, less memory space.

- If you want display images directly on the page then we use jpg

GIF - Only 256 colors, less pixel depth, more memory, animated.

- Always use for gif for buttons, logo, bullets, line etc.

SVG - Vector Graphics, not pixel based, high resolution and it is a xml based

Webp - It is image embedded into page, so that you cannot directly save image

TIFF - Not mandatory, used in documents

Bmp - used for Binary format image

ICO - It is used for icons [favicon]

Attributes:

1) src : It defines path and name of the page

Syntax:

`<img src = "image/pic.jpg">`

2) alt : It defines the text alternative text to display when image fails to load.

Syntax:

`<img src = "image/pic.jpg" alt = "please enable images on your browser">`

3) title : It defines the tooltip to display when mouse pointer is over image.

Syntax :

```
<img src = "images/pic.jpg" alt = "Image here..." title = "some tips about image">
```

4) width & height : It defines the dimensions of the image. Fluid images are defined by using dimensions specified with "%".

Syntax :

```
<img src = "images/pic.jpg" width = "800" height = "400%">
```

5) align : It is used to align the image left or right or centre by wrapping the text around.

Syntax :

```
<img src = "images/pic.jpg" width = "700" height = "400" align = "center">
```

6) vSpace & hSpace : It sets horizontal and vertical space between text image.

Syntax :

```
<img src = "images/pic.jpg" align = "right" vSpace = "40" hSpace = "40">
```

### Advanced Attributes:

1) CrossOrigin [CORS] - CROS - Origin Resource Sharing.

- used when loading images from another domain (like a CDN)

- controls whether to send credentials (cookies, HTTP auth) when fetching the image

Syntax :

```
<img src = "...cdn...path..." crossOrigin = "user-credentials | anonymous">
```

2) decoding : it defines how an image have to load along with another content in page.

it have the techniques : sync, async, auto.

Syntax :

```
<img src = "images/pic.jpg" decoding = "sync | async | auto">
```

3) importance : It sets priority for image , which can be low, medium, high, auto.

Syntax :

```
<img src = "images/pic.jpg" importance = "low | medium | auto | high">
```

4) srcset: It is used to display a set of images, which can be seen change randomly.

Syntax:

```
<img srcset = "array of Images">
```

## LINKS IN HTML

\* Link is a clickable Text, picture or graphic that navigates the user from one location to another.

, Links are also called as "Hyperlinks" in a web page.

- To create hyperlink html provides '' tag which stands for "anchor".

- Hyper links are classified into 2 types.

1) Intra Document Links

2) Inter Document Links.

i) Intra Document Links [Links within in the same HTML document (page)]

\* An Intra Document Links navigates user from one location to another location within the same page

Syntax:

```
<a href = "http://www.google.com"> Visit Google </a>
```

href → HyperText Reference

↳ It is an attribute to specifies the destination.

- Configuring intra document links comprises of 2 phases

→ attribute

1) defines ID for targeting element

2) Navigate the ID by using  href attribute

Syntax:

```
<h2 id = "html1">
```

```
<img id = "realme">
```

```
<a href = "#realme"> html1 </a>
```

FAQ1:

how to change color for visited link?

By using "vlink" attribute in <body> element:

```
<body vlink="green">
```

how to change color for active link?

By using "alink" attribute in <body> element:

```
<body alink="green">
```

how to change color for unvisited links?

By using "link" attributes in <body> element

```
<body link="green">
```

(or)

By using CSS styles:

```
a {  
    color: green  
}  
a:visited {  
    color: green  
}  
a:active {  
    color: green  
}
```

how to remove underline for hyperlink?

```
a {  
    text-decoration: none;  
}
```

a) Inter Documents Links [link between different HTML document (pages)]

- Inter Document Links link can be navigate to any file, url or application.

- You can configure a link that navigates to any specified file in project.

- It is a hyper link that takes you from one HTML document (page) to another HTML document (page).

Syntax:

```
<a href="page1.html"> Go to page2 </a>
```

## FAQ

- How to configure a link that downloads the given file?

By using "download" attribute for `a` element:

Syntax:

```
<a href = "images/pic.jpg" download = "filename">Fashion </a>
```

- How to open linked document in new window?

By using 'Javascript' window.open() method

Syntax:

```
<a href = "javascript : window.open('path', 'features')">Text </a>
```

```
<a href = "javascript : window.open("images/a1.jpg", "fashion", "width=300  
height=400")"> A photo </a> </p>
```

- How to Open linked document in the same page along with existing content?

By using `<iframe>` of HTML 5

It is the new HTML 5 element, which is used to embed any content into webpage.

It is used for displaying a nested webpage (a webpage within a page). The HTML

`<iframe>` tag defines an inline frame, hence it is called "inline frame".

Syntax:

```
<iframe src = "url"> </iframe>
```

We can use it for embed image, documents, audio, video etc.

We can use it for navigating to any specified application

Syntax:

```
<a href = "mailto: some@gmail.com"> Email </a>  
" " " skype: some@outlook.com > Skype </a>  
" " " tel: +91 910530041 > Call </a>
```

mailto → Open mail application

Skype → " Skype app

tel → " phone "

## Entities in HTML

A Text fragment (or "string") that starts with an ampersand (&) and ends with a semicolon (;) is called HTML entity.

- Entities are widely used to show unseen characters such as non-breaking spaces, Reserved characters, which would be read as HTML code. ✓

Ex: < - &#8377;  
nbsp - non-breaking space

|                        |              |
|------------------------|--------------|
| &lt; → < (less-than)   | %lt; ; → <   |
| &gt; → > (greaterthan) | %gt; ; → >   |
| &dollar; → dollar (\$) | 15 - 19 - 25 |
| &copy; → ©             |              |
| &trade; → trade        |              |

## Tables in HTML

- Tables are used to arrange the content in the form of rows and columns.
- There are different columns within a row, and each cell may contain any type of content.
- upto HTML4 tables are used for designing layout of page.
- From HTML5 tables are used for designing data grid.
- Data grid allows to handle CRUD Operations on data.
- The table cell can contain not only content but also a picture list, hyperlink or even another table.

## HTML Table Tags:

- 1) <table> : It defines a table
- 2) <tr> : It defines a row in a table
- 3) <th> : It defines a header cell in a table
- 4) <td> : It defines a table data cell (which means In table, cell having data)
- 5) <caption> : It defines a the table caption
- 6) <colgroup> : It specifies the group of one or more columns in a table for formating
- 7) <tbody> : It is used group the body content in table
- 8) <thead> : It is used group the header content in a table.

7) `<tfoot>` : It is used to group the footer content in table.

`<caption>` → student Table

| SNO | Name         | Age | Branch |
|-----|--------------|-----|--------|
| 1   | Shivam       | 20  | CSE    |
| 2   | Chiru        | 21  | ECE    |
| 3   | Gaurav Kumar | 22  | CSE    |
| 4   | Bhaskar      | 23  | ECE    |

### Forms in HTML

Form provides an UI from where user can interact with our application.

User can submit a query so that our application will process the query and generate response.

An HTML form is a section of document which contains controls such as as text fields, password fields, checkboxes, submit button, radio, listbox, dropdown etc.

It is the set collection of set of elements, which can be submitted to server.

Every web page can have multiple forms.

It is designed by using `<form>` tag.

Syntax:

`<body>`

`<form>`

... form elements ...

`</form>`

`</body>`

### Attributes:

- 1) Id : a unique identifier for an element
  - 2) name : identifies an element when data is submitted to the server
  - 3) value : used to group multiple elements under the same style/behaviour
- Note:
- `id, name, value` are used for Reference technique used to access a form in web
- 4) method : defines how the form data is sent to the server
    1. get → Appends data in the URL (visible in address bar)
    2. post → sends data in the Request Body (not shown in URL)
  - 5) action:
 

Defines the target URL/resource where form data will be submitted.

HTML5 have default validations, which you can disable it.

(required, type="email", pattern etc)

### Syntax:

```
<form method="post" action="page.jsp" novalidate>
</form>
```

### Form Elements

- Form Elements are the interactive controls inside `<form>` tag that let users enter and send the data to a server.
- They are the building blocks of user input in web pages.
- Every typical HTML form comprises of various elements like:

button  
textbox  
checkbox  
radio  
listbox  
dropdown etc.

element  
HTML provides different element for creating elements

```
<input>
<select>
<option>
<optgroup>
<datalist>
<textarea>
<progress>
<meter>
```

Note:  
all the elements must be the inside the page form container if elements are configured outside <form> then page cannot submit their data to server.

### HTML Forms Tags:

- 1) <input>  
  - <input> tag in HTML is used to collect user input in web forms.
  - It facilitates user to input data and communicate with a website or application
  - <input> have no end tag

### Syntax:

```
<input type = "value" />
```

### 2) <form>

- <form> tag is used to defines the form

### Syntax:

```
<form>
<!-- form elements -->
</form>
```

## 3) &lt;select&gt; :

- <select> tag is used to create drop-down list for user input, containing <options> tags to display the available choices.
  - It provides functionality for selecting one or multiple options from a list.
- Note: <select> tag is used in a form to reduce user responses.

Syntax:

```
<select>
  <option>
    </option>
    ...
  </select>
```

## 4) &lt;option&gt;

- <option> tag is used define individual items in a list

Syntax:

```
<option>
  contents ....
</option>
```

## 5) &lt;optgroup&gt;:

- <optgroup> is a tag which is used to group related <option> elements within a dropdown list
- Which specifies a label or heading for the group of options

Syntax:

```
<optgroup label = "Group Name">
```

```
  <option value = "Value 1"> Option 1 </option>
```

```
  <option value = "Value 2"> Option 2 </option>
```

```
  ...
</optgroup>
```

### 6) <datalist>

- <datalist> tag in html provide auto - complete suggestions to users
- saves typing time
- useful when options are many, but you still want to allow free text (eg: names, emails, passwords).
- When you entered the data in the form then it will auto suggests.

Syntax:

```
<input list = "datalist-id">
<datalist id = "datalist-id">
    <option value = "Option-1"></option>
    <option value = "Option-2"></option>
    ...
</datalist>
```

### 7) <textarea>

- <textarea> tag is used to define a multiple text input control
- It can hold unlimited number of characters and texts are displayed in a fixed-width font (usually Courier)

Attributes:

Syntax:

```
<textarea name = "text" rows = "10" cols = "80">
    Enter text here .....
</textarea>
```

1) name : name of the field (text.)

2) rows : Number of visible text lines

3) cols : width of the textarea (in character columns)

- ### (8) <progress> → how much of a task is completed.
- <progress> tag is used to display the progress of a task.
  - It provides an easy way for web developers to create progress bar on the website.
  - It mostly used to show the progress of a file uploading or downloading on the web page.

→ type → 0 → 100  
 Ex:- File uploading, drawing, loading etc

Syntax:

```
<progress max="100" value="50">
</progress>
```

Attributes:

a) max → It represents total work to be done to complete task.

b) value → It represents the amount of work that is already completed.

9) `<meter>`

- `<meter>` tag is used to measure data within in a given range.

- It defines a scalar measurement with range.

- It is also called Gauge

Ex: Battery %, Exam Score, Disk usage

Syntax:

```
<meter attributes ...>
```

Ex: `<meter value="5" min="0" max="100">`  
5 out of 10  
</meter>

</meter>

10) `<button>`

- `<button>` tag is used to create clickable buttons on a webpage

- These buttons can trigger various actions, such as submitting a form, summing, Javascript functions, or simply interacting with users

`<button>` tag contains text, images or other HTML elements

Syntax:

```
<button> Click Me! </button>
```

Note: We can also reset the button, but you have to "reset" attribute in type.

11) `<label>`

- `<label>` tag represents a caption for a form element in a user interface.

- It tells the user what the input is for like ("Name", "Password", "Email")

Syntax:

```
<label for="student">
```

Attributes:

a) for → It is an attribute which is used to specify the type form of element a label is bound to.

</label>

## Attributes of `<input type = "text">` [Text Box - Level]

### • Text Box:

textBox is used to handle the string input.

String is a literal with group of chars, which includes [A-Z, 0-9, special characters].

To make `<input>` as textBox you have to use the attribute "type".

### Syntax:

`<input type = "textbox">` → HTML 4 style

`<input type = "text">` → HTML 5 style

### Attributes:

1) id  
2) name } Used to Refer the element  
3) class

4) value: It defines the default value to display in text box

### Syntax:

`<input type = "text" value = "username">`

### 5) readonly:

It will not allow to modify value in the textBox, but it submits the values.

### Syntax:

`<input type = "text" name = "country" value = "Noida" readonly />`

### 6) disabled:

It will not allow to modify the value in textBox. It will not submit value.

- You cannot edit the value

- Value will not submitted in form

- Element is skipped during form submission

### Syntax:

`<input type = "text" name = "username" value = "jhon" disabled />`

### 7) placeholder:

- provides a hint to the user of what can be entered in the "input field".
- It is used to specify the expected value to displayed before user input element in the textbox.

Syntax:

```
<input type="text" name="username" placeholder="Enter your Username">
```

### 8) autofocus

It sets focus to the element automatically on page load

Syntax:

```
<input type="text" placeholder="name must be 10 char" autofocus>
```

### 9) required:

- It is used to ensure that textbox is having value
- It is used to allow to submit a blank value
- It is used to specify that the input element must be filled out before submitting form

Syntax:

```
<input type="text" name="username" required>
```

### 10) size:

- Syntax: `<input type="text" name="thru" size="4">`
- It defines the width for textbox; the default is 20
  - It determines the how many characters can be seen in the "input" field at one time.

### 11) minlength:

It validates the minimum number of char in textbox

Syntax:

```
<input type="text" minlength="10" placeholder="name min 10 char">
```

### 12) maxlength:

It is used to specify the maximum no. of characters entered into the textarea element

Syntax:

```
<input type="text" maxlength="15" placeholder="name max 15 char">
```

### 13) list:

It specifies the id <datalist> element that contains predefined options -for an

`<input>`

... comes with "text, search, url, tel, email, number"

Syntax:

```
<input type="button" list="Options">

<datalist id="options">
    <option value="java">
    <option value="python">
    <option value="javascript">
</datalist>
```

\*\*\*

#### 14) Pattern:

- pattern attribute is used to verify the form of input value.
- It is used with `<input>` elements to specify a Regular Expression that the input's value must be match for the form to submitted.

Syntax:

```
<input pattern="regular-Exp">
```

- format is about how it starts, ends and what it is contains etc.
- Patterns Requires "Regular Expression" to verify the format of input value.

A Regular Expression is built by using

- a) Meta characters
- b) Quantifiers

#### a) Meta characters

Characters

|         | description                                                               |
|---------|---------------------------------------------------------------------------|
| ?       | Zero or one occurrence of character<br>Ex: colour? => COLOR, Colour       |
| *       | Zero or more occurrence of character<br>Ex: colour* => COLOR              |
| +       | One or more occurrences of character<br>Ex: colour+ => COLOUR, colourY    |
| • (dot) | any single character<br>Ex: .at => cat, bat, sat, dat<br>..t => .cot, cat |

|                                                                 |                                                                                                                                                                                             |
|-----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                 | wlogical OR, any 1 of gives sets of value<br>ex: pattern : "green blue red"                                                                                                                 |
| ① - Starts with                                                 | It is an escape sequence character<br>it changes a normal character into meta character vice versa.                                                                                         |
| ② - Ends with                                                   | It defines excluding specified character starts with<br>ex: pattern = "Ad" → allow only "d" and exclude others                                                                              |
| ③ - It defines starts with                                      | It defines ends with<br>ex: pattern = "In" expression ≠ "                                                                                                                                   |
| ④ - It defines union of chars or expressions                    | It defines union of characters or multiple char set [collection]                                                                                                                            |
| ⑤ - It defines the Range of characters                          | Ex: pattern = "[a-d]" → a,b,c,d<br>" = "[a-a]" → all other than a-d<br>" "[a,t,u]" → only specify allowed<br>" "[a,f,w]" → all others allowed excluding "a, f, w".                          |
| ⑥ - any single digit number [0-9]                               | ex: pattern = "[0-9]"                                                                                                                                                                       |
| ⑦ - It indicates non-numeric (all characters other than number) | It indicates non-numeric (all characters other than number)<br>ex: pattern = "ID" → [a,z, A-Z] and special characters are allowed<br>ex: pattern = "1old" → A3<br>ex: pattern = "1a1D" → 3A |
| ⑧ - It Represents single blank space                            | Note: Never use directly a space in pattern always configure with "ls".<br>ex: pattern = "10ls1s" → A 4, \$5                                                                                |
| ⑨ - (small letters)                                             | It defines a word char<br>word char can be [A-Z, a-z, 0-9]<br>ex: pattern = " w" → [A-Z, a-z, 0-9]<br>ex: pattern = "lwlw" → [a4, -4, A5] ✓                                                 |
| ⑩ - (capital letters)                                           | only special chars are allowed other than - (underscore)<br>ex: pattern = " W" => #!@%_&%                                                                                                   |

Quantifiers → this define how many times a character, group or pattern must occur.

$\{n\}$  exactly  $n$  number of char

$\{n,m\}$  minimum -  $n$  & maximum -  $m$   
Ex: pattern = "1d{4,10}"

$\{n,\}$  minimum -  $n$  & maximum - any  
Ex: pattern = "1d{4, }"

Ques) Write the Regular expression to validate IFSC code SBIN000009345

input type="text" name="IFSC" pattern="SBIN000009345"

input type="text" name="mobile" pattern="1+91{10}"

Write a Regular Expression to validate 10 digit mobile number starting with +91.

input type="text" name="mobile" pattern="1+91{10}"

Write a Regular Expression for to validate US validate +1(415) 555-0100

input type="text" name="IFSC" pattern="1+(1)+(14{3}3)1s1d{3}-1d{4}"

- You can implement pre-defined Expressions.

C? = . \* [A-Z] atleast one upper case letter

C? = . \* [a-z] if lowercase

C? = . \* [0-9]

C? = . \* [!@#\$%^&\*]

- We can use CSS validation classes

:valid → applied when the input value matches the pattern

:invalid → applied when the input value does not match the pattern (or is empty if required is set)

## Types of Input Fields :

### 1) Password

- It allows a string input masked with password character "#".
- all attributes are same as textbox, only "list" cannot be used for passwords.

Syntax:

```
<input type = "password" name = "password" required minlength = "4" maxlength = "10"
       pattern = "[w{4,10}]>
```

### 2) Number

- It allows to input as a numeric value
- all attributes are same as textbox, only by have a range configured with

a) min → sets minimum allowed value

b) max → sets maximum allowed value

c) value → sets the default value (20)

d) step → sets the increment / decrement interval

Syntax:

```
<input type = "number" max = "30" min = "13" step = "3" name = "Age"
       ↳ which will increase 3, 6, 9, 12 ..
```

### 3) Range

It is similar to number input but allows user to select numeric value using slider

Syntax:

```
<input type = "range" min = "1" max = "1000" step = "5" name = "Qty">
```

### 4) Email

- It validates the email address
- It is just verifying the availability of "@" in a string.

Syntax:

```
<input type = "Email" name = "email" required placeholder = "your Email address">
```

## 5) URL

- It validates URL format for input value
  - URL must have a protocol and domain extension
- Ex:- `http://protocol : //www.amazon.in` → valid  
`www.amazon-ion` → invalid

Syntax:

```
<input type = "url" name = "myurl"
```

## 6) File

- \* It is a file picker, that allows user to browse and select a file from their device

\* It is not a file uploader, it is a file selector -  
 When we want file upload we will use javascript functions upload function

syntax: no filters

```
<input type = "file" name = "photo"
```

syntax: single filter `accept = ".jpg"`

syntax: multiple filters

```
<input type = "file" name = "photo" accept = ".jpg, .gif, .png"
```

- \* If you want multiple files to choose then you can define "multiple" attribute

syntax: `accept = ".jpg, .png" multiple`

`<input type = "file", name = "photo" accept = ".jpg, .png" multiple>`

- \* File selector allows to choose only one file

## 7) Radio Buttons

- Radio Buttons allows user to select one or more (or multiple option) a group of choices

• Radios button once checked and it cannot be unchecked

- Radio's can be MULTEd MAXIMU

Syntax:

```
<input type = "radio" name = "gender"> Male
```

```
<input type = "radio" name = "gender"> Female
```

- You can make the radio selected by default by using "checked" attribute

```
<input type = "radio" name = "gender"> Male
```

```
<input type = "radio" name = "gender" checked> Female
```

### c) Date & Time

- Date & Time provides types for date and time input
- It provides users with an easy-to-use calendar and choose specific date.

- date
- datetime-local
- time
- month
- year
- week

\* You can configure the date picker with restricted date and time range, by using below attributes.

- min
- max

\* The date & time for time format developer is:

"yyyy-mm-dd hrs:min:seconds.milliseconds"

< syntax :

```
<input type = "date">
```

```
<input type = "time">
```

```
<input type = "datetime-local">
```

```
<input type = "month">
```

```
<input type = "week">
```

```
<input type = "year">
```

Syntax: for Restricted [the date and time which means should select particular type]   
`<input type="date" min="2023-09-31" max="2023-10-23" name="departure">`

9) text  
 It facilitates short, straightforward text input for capturing simple information like names or comments

Syntax:  
`<input type="text" name="fullname">`

10) Color  
 It displays a color picker, that allows user to pick a color.

\* You can configure default color by using only 6 char hex code;

Syntax:  
`<input type="color" value="#ff0000">`

11) hidden

Operating in the background, this type especially manages and stores data without user visibility

Syntax:  
`<input type="hidden" name="user_id" value="737"`  
 (In browser it will not show and automatically stored data with id 737 ✓)

12) image

This type acts as a clickable image, sometimes serving as a visual trigger for form submission

Syntax:  
`<input type="image" name="photo" src="http://media.greengrocer.com/images/50x301.jpg" height="50" width="301">`

13) reset

It creates a button that resets all form fields to their default values  
 It only resets to original state

Syntax:

`<input type="reset">`

## 14) Search

It is used to search boxes, this type encourages user to type and initiate search operations.

Syntax:

`<input type = "search" />`

## 15) tel

It is designed for telephone numbers, this type streamlines the input process with numeric keyboards.

## 16) button

This type serves as a clickable button, often used for custom interactions.

Syntax:

`<input type = "button" value = "Click Me!" />`

## 17) submit

This creates a button that submits the form data to server.

Syntax:

`<input type = "submit" value = "Submit Form" />`

## 18) checkbox

It is similar to Radio but allows user to check and uncheck the option + you can use for selection one or multiple.

Syntax:

`<input type = "checkbox" name = "Course" value = "java" checked > java`

Note: Checkbox submits "on" as value if "value" attribute is not defined.

FAQ: How to design a checkbox list?

HTML don't have any checkbox list.

You have to design manually by CSS overflow

CSS "checked" class:

checked is a class which verifies the whether the checkbox is checked or not.

If it is a new class in CSS used to verify the checked status of any checkbox or radio.

<input:

checkbox: checked ?

}

### 19) Buttons

- Button is used to confirm user actions.

#### a) Generic Buttons

Generic buttons are configured with pre-defined functionality.

HTML 4 Generic buttons

<input type = "submit" >  $\Rightarrow$  value is RC datatype

<input type = "reset" >

<input type = "button" > text | image

<input type = "button" > text | image | </button>

Note: if button type is not defined HTML button, then the default type is submit.

<button> click </button>  $\Rightarrow$  Submit

#### b) Non-Generic buttons

which means buttons are not configured with any pre-defined functionality.

##### a) HTML 4

<input type = "button" > value = "print" >

##### b) HTML 5

<button type = "button" > text | image | graphical </button>

## Form Controls

### 1) Dropdown

Dropdown, commonly known as <select> elements.

It allows us to choose one or more options from a predefined list.

Syntax:

```
<select>
  <option value = "Option 1"> Option 1 </option>
  <option value = "Option 2"> Option 2 </option>
  <option value = "Option 3"> Option 3 </option>
</select>
```

: every option comprises of 3 attributes

a) selected : It keeps the option selected on page load

b) disabled : It will not allow to select any option

c) value : the actual data sent data to the server when form is submitted

Syntax

```
<option selected> Delhi </option>
<option disabled> Mumbai </option>
<option value = "USA"> America </option>
```

NOTE:  
 . every <select> and <option> element is a `REData type`, you cannot present images or  
 symbols.  
 . It is only for plain text and special characters and it will not allow markup "inside option".

a) List Box

. It allows one or more multiple options from a group of choices.  
 . a dropdown is transformed into list by using following attribute.

a) multiple → Allows multiple choices the one option

b) size → how many options are visible at once

Syntax:

```
<select>
<select size = "2">
  <select multiple>
    </select>
```

### 3) Textarea

- It is used for multiple line Input
- text box allows only single line input, but textarea is required to accept input in multiple lines.

Syntax:

```
<textarea name="message" rows="5" cols="40"></textarea>
```

↓                                            ↓  
visible no. of lines (height)    width of the terms of any character count

- placeholder → Show a text inside the box until the user types
- readonly → we cannot edit, but can copy the text
- disabled → we cannot edit, but value wont be submitted.

### 4) Meter

- Meter is the new element in HTML5
- It is used display the grade meter in webpage

Attributes:

- min: minimum value of Range (default=0)
- max: maximum value of Range (default=1)
- value = current value (must b/w min & max)
- high = Range considered is High
- low = Range considered is Low

Syntax:

```
<meter min="1" max="100" low="0" high="100" value="100"></meter>
```

Note

if low and high are 0 then color → Green

if low and high difference is less than color → Red

if low and high difference is more than color → Yellow ✓

if low and high difference is more than color → Yellow ✓

### 5) Progress

It is used to show status of any task performed message in page, which including downloading, uploading, installing etc.

Syntax:

```
<progress value="100" max="100" aria-multiline="1"></progress>
```

### 1) Label

- It's used to display captions for elements in page
- Label improves accessibility (especially for screen readers)

Two ways to use <label>

#### 1) Implicit (wrapping method)

Syntax:

```
<label>
  Username:
    <input type="text" name="username" placeholder="Enter your username" />
</label>
```

#### 2) Explicit (Binding with for)

Syntax:

```
<div>
  <label for="username"> Username </label>
```

```
<div>
  <input type="text" id="username" name="username" placeholder="Enter your name" />
</div>
</div>
```

### Multimedia in HTML

Multimedia refers to audio, video, media animations.

### 1) Marquee

It is used to set scrolling content in page.

Syntax:

```
<marquee>
```

... your Content ...

```
</marquee>
```

Attributes:

1) scroll amount : speed of scrolling (1 to 100)

2) direction : Controls scroll's direction (Right, left, up, down)

3) behaviour : changes scrolling to sliding or type of movement

scroll , slide  
default , stop after one pass

alternate

Bounces back and forth.

4) loop: No. of times to repeat (default is infinite)

5) width & height : it sets width and height for marquee area.

6) bgcolor: Set background color.

Syntax:

`<marquee direction="left" behavior="alternate" scrollamount="5" loop="3">`

`</marquee>`

NOTE:  
You can pause and resume Marquee by using javascript functions  
`onmouseover="this.stop()" onmouseout="this.start()"`

## 2) Video

• `<video>` Tag is used in HTML to embed video files (`.mp4, .webm, .ogg`) directly into a webpage

- It allows user to play, pause, control volume, fullscreen etc.

Syntax:

`<video src="movie.mp4" controls></video>`

Attributes:

1) src → specifies the path/URL of the video file.

2) controls → displays pause, control, plays, volume, fullscreen controls.

3) autoplay → starts playing automatically (often requires muted)

4) loop → replays the video once its ends.

5) muted → start the video without sound

6) poster="`image.jpg`" → displays the image before the video starts (like a thumbnail)

7) width/height → sets the size the video player

3) audio:

• `<audio>` tag is used to embed sounds or music files (like `.mp3, .wav`) directly into web page.

• It allows the user to play/pause audio with controls.

Syntax

`<audio src="song.mp3" controls></audio>`

Note:

Attributes are same as like video.

Comments in html

- To add notes/ documentation in an html file ✓  
In Html, comments are special notes in the code that do not appear on the webpage
- they are meant for developer to explain code, leave, reminders or temporarily disable part of code.

Syntax

<!-- this is a comment -->

→ Hakon Wium Lie in 1994

16-9-2025

## CSS (styles)

\* CSS stands for Cascading Style Sheets

\* CSS is a widely used language on the web

\* CSS helps web designers to apply styles to HTML tags

\* CSS utilized to make a responsive website

→ It describes to the user how to display HTML elements on the screen in proper format.

### Applications of CSS:

- 1) Styling Web pages
- 2) Responsive design → make websites adapt to different screen sizes [desktop, mobile]
- 3) Animations and Transitions
- 4) Website Management
- 5) Accessibility Improvement
- 6) E-commerce
- 7) Social Media

### CSS Versions:

- CSS 1 → Dec 17 1996 (Basic style properties for font)
- CSS 2 → May 12 1998 (display properties, pseudo classes, pseudo elements)
- CSS 3 → 1999 (selectors, Box model, Borders, backgrounds, grid layout)
- CSS 4 → upcoming (new features?)

### Types of CSS:

- 1) Inline CSS
  - 2) Internal CSS
  - 3) External CSS
- 1) Inline CSS
- styles are applied directly inside an HTML element using the style attribute.
  - The Inline CSS has the highest priority.

Syntax:

`<h1 style="color:red; font-size:18px;"> this is Inline style paragraph </h1>`

### a) Inline CSS (Embedded CSS)

styles are written inside the `<style>` tag in the HTML `<head>` section

Syntax:

```
<head>
  <style>
    body {
      background-color: green;
    }
    h1 {
      color: red;
      text-align: center;
    }
  </style>
</head>
<body>
  <h1> Internal CSS Example </h1>
</body>
```

### b) External CSS: (Recommended)

styles are written in a separate .css file and linked to HTML using the `<link>` tag.

Syntax:

index.html

```
<head>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1> External .css Example </h1>
</body>
```

style.css

```
body {
  background-color: #f0f0f0;
}
h1 {
  color: blue;
  text-align: center;
}
```

## Faq)

i) How to add CSS file to HTML by External CSS?

Ans i) Create a file with .css extension eg; style.css

a) Write your CSS Rules inside it

b) Links the CSS file in your HTML

→ In head section, use <link> tag

Ex:- <link rel="stylesheet" href="style.css">

ii) How to add CSS file to HTML?

\* In 3 ways

a) Inline CSS

b) Internal CSS

c) External CSS

CSS Selector :

\* CSS Selectors are used select the content you want to style.

\* Selectors are the part of CSS Rule set.

Types of Selectors :

1) primary Selectors

2) Pseudo Selectors

3) dynamic Selectors

4) Universal Selectors

5) Element state pseudo

6) Validation states pseudo

7) Structural pseudo

8) Root, languages selectors.

9) primary Selectors.

a) Type Selector

selects by element / Tag Name (P, h1, body etc.)

Syntax:

P { color: green }

- b) ID Selector → ID selector is defined with "#"
- Selects the element with a unique ID
- It selects a single, unique element
- Every element can have only one ID Reference

Syntax:

```
#header
{
background: gray;
}
```

- c) Class Selector:
- A class selector is defined by using ".classname" or ".classname"
- class selector is defined with ".(dot)"
- It is reused and rendered using "class" attribute.
- Every element can implement multiple classes.

d) Universal Selector

- the universal selector is used as a wildcard character.
- It select all the elements.
- Universal selector is defined with "\*".

Syntax:

```
* {
color: green;
}
```

e) Group Selector:

- the group selector is used to select all the elements with same style definitions
- Group selector is used to minimized code. ,(Comma) are used to separate each selector in grouping

Syntax:

```
h1, h2, p, body {
text-align: center;
color: green;
}
```

f) Pseudo-selectors (Combinators)

These define relationships between elements.

1) parent child: child selector, it can access all child elements under specific parent from any level

Syntax:

ul li {

}

nav div {

}

2) parent>child: direct child selector, It cannot access if elements is at multilevel hierarchy

Syntax:

nav>div {

}

3) General sibling: It can access all elements that are descended after specific element. It is configured with "~"

Syntax:

A~b {

}

4) Adjacent sibling: It can access the only adjacent element and It is configured with "+".

Syntax:

A + B {

}

5) structural pseudo class:

used for selecting elements based on their position in DOM.

: first-child → first child element

: last-child → last child element

:nth-child() - specific child element you can define even or odd  
 :nth-child(n) - nth child of parent.

li: nth-child(3) → 3rd li /

{color: Red}

li:nth-child(odd) {color: blue}

li:nth-child(even) {color: white}

:nth-of-type(n) → nth child of a specific type

Syntax  
 p: nth-of-type(2) → 2nd <p>

{color: green}

(last li) ✓

:nth-of-type(1) →

{color: orange}

### 3) Attribute Selectors

[attr] → elements with attribute

[attr="value"] → exact match

[attr\*= "value"] → starts with

[attr\$= "value"] → ends with

[attr\*="value"] → contains anywhere

[attr\$="value"] → starts with "value" or "val-

[attr~="value"] → contains word

### Syntax:

input [type="text"] {color: green}

a [href="https"] {color: blue}

img [alt~="profile"] {color: red}

## 8) Behaviour pseudo classes:

- ::before → insert Content before element
  - ::after → insert Content after element
  - ::first-letter → style the first letter
  - ::last-letter → style the last letter
  - ::placeholder → style the placeholder text
  - ::first-line → style the first line
  - ::selection → style highlighted text
- :: → Inheritance

## 9) Element state pseudo classes

a) :enabled → Targets form elements that are enabled (layout size)

Syntax

```
input:enabled
{
    color: green
}
```

b) :disabled → When the elements are disabled

Syntax

```
input:disabled
{
    color: green
}
```

c) :checked → Matches Radio buttons or checkboxes are checked

Syntax:

```
input [type = "checkbox"] :checked {
    outline: 2px solid green }
```

d) :required → matches inputs with the required attribute

Syntax:

```
input:required {
    border: 2px solid green }
```

- > e) :focus → effects on element's focus
- > d) :blur [obsolete - no longer in W3C] → effects when element lost focus.
- > f) :read-only → when element set to read-only
- > g) :option → matches with input "required" attribute.
- > syntax:
- > input:optional {  
border: solid gray  
}
- > h) valid: → matches the input field whose value meets validation rules (constraint)
- > syntax:  
input:valid {  
border: 2px solid green }
- > i) invalid → matches the input whose value does not meet validation rules
- > syntax:  
input:invalid {  
border: 2px solid gray }
- > j) in-range and out-of-range  
works with input that have min/max values
- > input [type="number"] :in-range { border: 2px solid red }
- > input [type="number"] :out-of-range { border: 2px solid green }

#### 10) Dynamic Pseudo Classes:

- dynamic allows to change according state and situation.

:hover → It defines action on mouse over

:active → It defines onmousedown (hold mouse down button)

:link → It defines actions for hyperlink in normal state.

:visited → ↗ effected for visited links

:target → ↗ " a for link when it is target.  
[Elements becomes target on link click]

### Comments in CSS:

\* Comments are used in CSS (Cascading style sheet) to add clarifications or reminders to the code or document to the code that ignores by the browser.

#### Syntax:

/\* this is CSS single line comment \*/

/\* this is a CSS  
multi-line comment \*/

### Styles with CSS

#### a) Colors

In CSS, you can set colors for text, backgrounds, borders, margins etc.

##### Color formats in CSS

###### a) Color names

CSS supports 140+ predefined color names

Syntax: Ex: Red, blue, white, yellow, Green, pink, orange, black, violet etc

H1 {

color: red; }

P {

background-color: Blue; }

###### b) RGB

Each value = 0 to 255

Syntax:

P {

color:rgb(255,0,0); } // → Red

## c) RGBA

same as RGB but with alpha (opacity)  $\rightarrow$  0 transparent to 1 (opaque)

Syntax: rgba  $\rightarrow$  red, green, blue, alpha  $\rightarrow$  alpha range from 0 to 1

h1 {  
background-color: rgba(0,0,0,0.5); }

## d) Hexadecimal Code

Written #RRGGBB (00-FF each color)

Syntax:

h2 {  
color: #f00; } // red

e) HSL (hue, Saturation, Lightness)  
• hue  $\rightarrow$  degree on color wheel (0 - 360)

0 - 120  $\rightarrow$  Red shades

120 - 240  $\rightarrow$  Green shades

240 - 360  $\rightarrow$  Blue shades

• saturation  $\rightarrow$  intensity (0% = gray, 100% = full color)

• lightness  $\rightarrow$  brightness (0% = black, 50% = normal, 100% = white)

Syntax:

h3 {  
color: hsl(120, 100%, 50%); } // green

d) Gradient colors  $\rightarrow$  smooth blending for multiple colors  
Configured with background-image

## a) Linear Gradient

body {  
background-image: linear-gradient(to right, red, green 50%, yellow); }

## b) Radial Gradient

div {  
background-image: radial-gradient(circle, red, yellow, green); }

## II) Background properties

CSS Background properties are used to define the background effects on element.

- 1) background-color → to specify the background color of the element
- 2) background-image → adds an image as background
- 3) background-position → sets the starting position of the background image
- 4) background-repeat → defines how the background image repeat
- 5) background-attachment → defines the background image scrolls with page  
(scroll, fixed, local)

Syntax:

```
body {
    background-color: green;
    background-image: url("bg.jpg");      (or)
    background-position: centre top;
    background-attachment: fixed;
    background-repeat: no-repeat;
}
```

```
fixed centTop;
background: green
" url("bg.jpg") no-repeat;
```

## III Text styles

1) Color: set the text color

```
p {
    color: green;
```

```
}
```

2) text-align: Align text horizontally  
values = Left / Right / center / justify

Syntax:

```
m
{
    text-align: center;
}
```

3) text-decoration : Adds or removes decorations from text  
 Values = none / underline / overline / line-through

Syntax:

```
a {
  text-decoration: none;           // Removes underline from link !
}
```

4) text-transform : Change case of text

Values : uppercase / lowercase / capitalize

Syntax:

```
p {
  text-transform: uppercase;
}
```

4) text-indent : Add space at the beginning of the first line of text

Syntax:

```
p {
  text-indent: 50px;
}
```

5) letter-spacing : control spacing between characters.

```
h2 {
  letter-spacing: 3px;
}
```

## IV) Font Properties

\* Font properties is used to control look of texts.

\* By the use of font properties we can change text size, color, style etc.

1) Font color

Changes the color of text

Syntax:

```
p {
  color: red;
}
```

#### 4) font-family

- Defines which font face to use.

Syntax:

```
p
{
  font-family: Arial, Helvetica, sans-serif;
}
```

#### 5) font-size:

- Sets the font size.

Syntax:

```
h1
{
  font-size: 32px (or) 12px;
}
```

#### 4) font-style

- Defines if text should be normal, italic, oblique etc.

Syntax:

```
p
{
  font-style: normal;
}
```

#### 5) font-weight

- Control how bold or light text is.

Syntax:

```
h1
{
  font-weight: bold;
}
```

## IV) Border properties

A border in CSS is a line drawn around HTML element content + padding area

### 1) border-style:

- Sets the type of line for the border
- Without this, the border won't show (style required)

Syntax:

```
div
{
    border-style: solid;
}
```

Values:

- none → no border (default)
- solid → single solid line
- dotted → dotted line
- dashed → dashed line
- double → two lines
- groove, ridge, inset, outset  
→ 3D effects.

### 2) Border-width

- Defines the thickness of the border

Values:

- Keywords → thin, medium, thick
- Units → px, em etc.

Syntax

```
div
{
    border-width: 5px;
}
```

or

```
div
{
    border-width: 5px 15px 10px 2px;
}
↓   ↓   ↓   ↓
Top Right bottom left
```

### 3) border-color:

- Sets the border's color

Syntax:

```
div
{
    border-color: red;
}
```

or

```
div
{
    border-color: red green blue yellow;
}
```

↓ right top, right, bottom, left

### 4) border-radius:

- Rounds the corners of the border

Syntax :

div

{

border-radius: 15px;

}

### V. Margin Properties:

- Margin properties controls the space outside the border of an element
- It is transparent spacing color:

Syntax :

p

{

margin-right: 15px; → sets the margin on the right side of element

margin-top: 15px; " " " on the top of element

margin-left: 15px; " " " margin on the left side

margin-bottom: 15px; " " " on the bottom of element

}

### VI padding:

- padding is the space between the content of an element and its border

#### 1) padding:

Applies padding all sides (top, right, left, bottom)

Syntax:

div

{

padding: 20px;

}

## 2) padding-top:

Add the padding above the content

Syntax:

```
div {
    padding-top: 10px;
}
```

## 3) padding-right:

Adds the padding to the right of corner of the content

Syntax:

```
div {
    padding-right: 10px;
}
```

## 4) padding-bottom:

- Add the padding bottom of the content

Syntax:

```
div {
    padding-bottom: 10px;
}
```

## 5) padding-left:

Adds the padding on left of the content

Syntax:

```
div {
    padding-left: 15px;
}
```

short form →

padding: 15px 10px 15px 25px;  
top right bottom left

CSS units or measurements in CSS:

### 1) px (pixels)

- Absolute unit (fixed size)
- mostly used in borders, small spacing etc.

### 2) % (percentage)

- Relative unit (dependent on parent element)
- common for width, height, padding, margin

### 3) vh | vw (viewport height | viewport width)

- Relative to the viewport size
- useful for full-screen, hero banners, layouts

Syntax:

```
px {  
    height: 100vh;  
    width: 100vw;
```

### 4) fr (fractional unit in CSS Grid)

- Represent the portion of available space in grid layout

Syntax

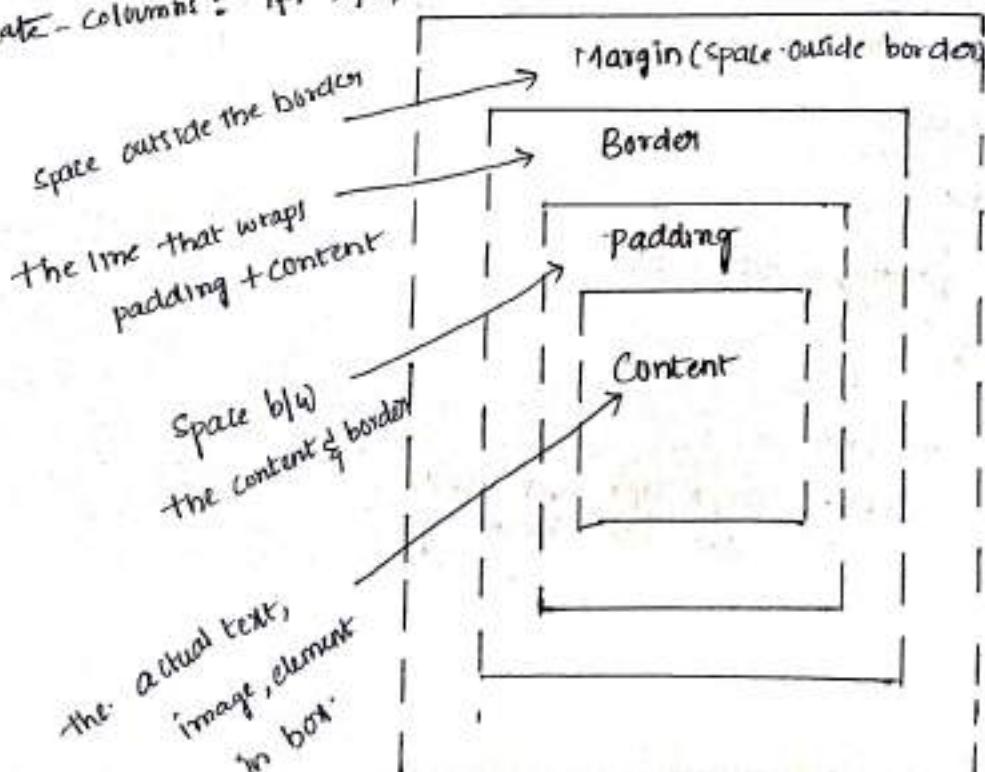
- containers

```
display: grid;
```

```
grid-template-columns: 1fr 2fr;
```

```
}
```

Box Model:



## (VIII) Display properties

\* It determines how the element is displayed in the layout.

- \* It specifies the how the element is displayed in the layout.

1) display: inline

- \* To display the element "inline". It does not start a new line and respect the content flow.

Syntax:

```
span {
    display: inline;
}
```

2) display: block

- \* the element takes full width, starts on a new line

Syntax:

```
span {
    display: block;
}
```

3) display: inline-block

- \* Behaves like inline, but allows width and height to be set.

Syntax:

```
img {
    display: inline-block;
}
```

4) display: none

- \* the "none" value totally removes the element from the page

- \* It will not take any space

Syntax

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

## X1. Flexbox Properties

Flexbox = Flexbox Layout → helps to align and distribute space in a container.

Syntax:

```
• container {
    display: flex;           // Enabling
}
}
```

### a) Flex Container Properties

a) display: flex → defines a flex container

Syntax:

```
• container {
    display: flex;           // Enabling
}
}
```

### b) flex-direction:

We can arrange the element in any direction, like row, column or a reverse column, a reverse row.

Values: row | column | row-reverse | column-reverse

Syntax:

```
• container {
    flex-direction: row;     // default
}
}
```

### c) flex-wrap

It specifies the whether or not flex items should be wrap across multiple lines or remain on single line.

Values: nowrap | wrap | wrap-reverse

Syntax

```
• container {
    flex-wrap: wrap;
}
}
```

## d) flex-flow

combines flex-direction + flex-wrap

Syntax:

```
. container {
    -flex-flow: row wrap;
}
```

## e) justify-content:

aligned horizontally  
It determines how flex items are aligned inside the container  
values: flex-start / flex-end / space-between /  
space-around / space-even-distribution.

Syntax:

```
. container {
    justify-content: flex-start; /* default */
}
```

## f) align-items

Defines the vertical alignment of flex items inside the container.

Syntax:

```
. container {
    align-items: flex-start;
}
```

## g) Flex Item properties

## a) Order

control order of items

item

```
. item {
    order: 2; /* default is 0 */
}
```

## b) flex-grow

Defines how much items grows if space available

Syntax:

```
. item {
    flex-grow: 1;
}
```

### c) flex-shrink

defines how much items shrinks when not enough space

### • item 5

flex-shrink: 1

}

### d) flex-basic

define initial size of item

### • item 5

flex-basic: 200px;

}

## X Grid System Properties

Grid Layout allows to divides the webpage into the rows and columns

### Grid Container Properties

#### a) display: grid

It can define the element as the grid container

Syntax:

• container

{

display: grid;

}

#### b) grid-template-columns:

It can specifies the number and size of the columns in the grid

Syntax:

• container

{

display: grid;

grid-template-columns: 200px 200px 200px; / 3 fixed columns/

grid-template-columns: 1fr 1fr 1fr; / 3 equal columns/

}

c) grid-template-rows

It can specifies the the number and size of the rows of the grid

Syntax:

• container

{

display: grid;

grid-template-rows: 100px 15px;

}

d) grid-gap

It can specifies the spacing b/w the Rows and Columns

Syntax:

• container

{

display: grid;

grid-gap: 10px;

}

Note:

We can use fr unit (fraction unit) instead of pixel(px).

e) Positional properties

The position properties defines the how an element is placed in the document flow.

a) position: static

\* Default

\* positionally normally

\* top / left / right / bottom do not work

Syntax:

• box {

position: static;

}

### b) position: relative

- \* position relative to its normal position
- \* top | left | bottom | right "shift it, but it still takes original space"

Syntax:

- box {  
position: relative;

}

### c) position: absolute

- the element is removed from the normal flow (other elements act if it not there)
- position relative to its nearest positioned ancestor
- If no ancestor is positioned, it will be relative to the page

Syntax:

- container

{

position: absolute;

}

### d) position: fixed

- position relative to the viewport (browser window)
- stays in place if you scroll

Syntax:

- navbar

{

position: fixed;

}

### e) position: sticky

- A mix of Relative + fixed

- header {

position: sticky;

}

## XI Overflow properties

- the Overflow property controls what happens when content is too big for its container

### a) Overflow: visible (default)

\* Content is not clipped and is visible outside of element box.

Syntax:

```
.box
{
    overflow: visible;
}
```

### b) Overflow: hidden

- the overflow is clipped and rest of the content is invisible.
- No scrollbars are shown

Syntax:

```
.box
{
    overflow: hidden;
}
```

### c) Overflow: scroll

- Content is clipped but a scrollbar is added to see the rest of the content.
- the scrollbar can be horizontal or vertical

Syntax:

Overflow: scroll;

overflow-x → horizontal

overflow-y → vertical

### d) Overflow: auto

- Content is clipped
- It automatically adds a scrollbar whenever it is required (smart choice)

Ex:- overflow-x: hidden;  
 overflow-y: scroll;

Syntax:

```
.box
{
    overflow: auto;
}
```

## Bootstrap Framework

- \* the Bootstrap is a popular front-end framework used to quickly design responsive, mobile-first websites and web applications.
- \* It includes ready-made CSS and JavaScript classes for layout, design and components.

- \* Latest version: Bootstrap 5

Why we used Bootstrap?

- save time (no need to write CSS from scratch)
- makes responsive websites easy
- works with any Backend (Spring Boot, Node.js, Django, etc.)

How to add Bootstrap?

→ 1) Open cmd  
 2) Type "npm install bootstrap --save"  
 3) Link href = ".../node\_modules/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet"

Bootstrap classes:

① Container classes

- a container is the main wrapper for content in Bootstrap
- It provides fixed width or height or full width (fluid) layouts.

② Container

- Filled with containers
- main class in the container classes

Syntax:

```
<div class="container">
```

```
</div>
```

### b). Container - fluid :

- \* Always 100% width of the viewport
- \* Good for responsive

### c) .Container-sm: (small)

- \* Container small is class that styles a container that has 100% width till 540px, once the width is above 540px it will have fixed width.

Syntax:

```
<div class = "container-sm">
</div>
```

### d) .Container-md: (medium)

- \* Full width up to 720px , fixed width above 720px

\* Good for tab

Syntax:

```
<div class = "container-md"></div>
```

### e) .Container-lg: (large)

- \* Full width up to 960px , fixed width above 960px

\* Good for pc

### f) .Container-xl: (Extra large)

- \* Full width up to 1140px , Fixed width above 1140px

\* Good for Laptop

### g) .Container-xxl: (Extra Extra Large)

- \* Full width upto 1320px , fixed width above 1320px

\* Good for laptop wide screen

Syntax:

```
<div class = "container-*"></div>
```

Note:

Here, \* will be replaced by class values( sm, md, lg, xl, xxl )

### 2) Margin classes

\* Margin classes are used to add space outside an element

- a) .m → margin all sides
- b) .mt → margin on top
- c) .mb → margin on bottom
- d) .ms → margin on start/left
- e) .me → margin on end/right
- f) .m-size → size can be 1 to 5

#### size:

- 0 → no margin
- 1 → very small
- 2 → small
- 3 → medium
- 4 → large
- 5 → Extra large

Ex:-

M-4

Syntax:-

```
<div class="mb-4"> </div> // Bottom margin size 4 //
```

### 3) Padding classes

\* padding classes are used to add space inside element (between content and border).

- a) .p → padding on all sides
- b) .pt → padding on top
- c) .pb → padding on bottom
- d) .ps → padding on start/left
- e) .pe → padding on end/right
- f) .p-size → size can be 1 to 5

Syntax:

```
<div class="pt-3"> </div> // padding on top size 3 //
```

### 4) Border classes

\* Border classes are used to add borders, change border color, thickness, shape.

\* useful for boxes, forms, buttons, layouts formats.

- a) .border → adds a 1px border around the element
- b) .border-size → changes the border thickness (1, 2, 3, 4, 5)
- c) .border-0 → removes the border

d) .border - {color or contextually} → changes the border color

Note: primary → Blue

secondary → Gray

success → Green

danger → Red

warning → Yellow

Syntax:

<div class = "border p-3 border-success"></div>

e) .rounded → slightly Rounded CORNERS

f) .rounded-circle → perfect circle

g) .rounded-pill → Fully Rounded

h) .rounded-fsize{ } → Increasing the rounding size

i) .rounded-top / bottom / start / end → Rounds only top corners

Syntax:

<div class = "border rounded-circle p-3"></div>

5) Width and Height classes

\* Width and height classes are used to control the size of elements

\* useful for images, buttons, forms, boxes, div etc.

a) w-{size} → 0, 25, 50, 100

b) h-{size} → 0, 25, 50, 100

c) w-auto → width adjust automatically

d) h-auto → height adjust automatically

6) Buttons classes

\* Buttons are used for actions, navigation, form, and interaction

a) .btn : base class button

b) .btn - {contextually} : primary, danger, warning, secondary, success

c) .btn-link → looks like a link

d) .btn-sm → small button

e) .btn-lg → Large button

g) + btn - close → close button

h) + btn - group - vertical → Buttons are in vertical

i) + btn - group - horizontal → Buttons are in horizontal

j) + btn - toolbar → Group multiple group together in a single horizontal line

Syntax:

`<div class = "btn-primary"></div>`

2) Text and Background classes:

a) .text - {contextually} → primary, green, success, warning, danger  
(Changes the text color)

b) .bg - {contextual} → primary, green, success, warning, danger.  
(Changes the Background Color)

c) .text - {Alignments} → center, start, end

Ex:- Syntax:

`<div class = "text-center"></div>`

2) `<div class = "text-sm-center"></div>` // small text in centre.

d) .text - uppercase → text with uppercase

e) .text - lowercase → text with Lowercase

f) .text - Capitalies → text with Capital

g) rows and columns

\* row is used to create a horizontal group of column

\* It ensures proper alignment & spacing.

Syntax:

`<div class = "row"></div>`

\* col is used to create responsive, equal width column.

\* Bootstrap uses 12-column grid system

Syntax:

```
<div class = "row">
  <div class = "col-4" > column (4 parts) </div>
  <div class = "col-8" > columns (8 parts) </div>
```

9) Display content centred on screen:

- a) To center content both horizontally and vertically, uses Flexbox utility in Bootstrap
- b) .d-flex → makes the container a flex container
- c) .justify-content-center → centres content horizontally
- d) .align-items-center → centers content vertically

10) Form Classes

- a) .form-label → styles the label element
- b) .form-control → standard input for Email, text, number, url, password, date, textarea
- c) .form-control-color → standard input for color
- d) .form-control-file → standard input for file
- e) .form-select → dropdown/select menu
- f) .form-checkbox-input → for checkbox or radio button
- g) .form-check-label → label for checkbox / Radio
- h) .form-switch → container for switch-style checkbox

Syntax:

```
<div class = "form-switch" > </div>
```

11) Form Input Group classes

- a) .input-group → wrap the input and elements together
- b) .input-group-text → text or label before/after input
- c) .input-group-sm → small input group
- d) .input-group-lg → large input group

- \* Minification is a coding technique used by developers to compress code and reduce file size without affecting its functionality.
- \* It improves performance and reduce load time.

Ex:- /& Original CSS \*/

```
h1 {
  color: Green
}
```

/& Minified CSS \*/

```
h1 {color: #0f0}
```

Q) What is CDN?

CDN is (Content Delivery Network) a distributed network of servers that hosts libraries like (CSS, JS).

CSS Rules and priority:

FAQ)

If styles are defined all 3 techniques (inline, Embedded, External):

- If different properties → All apply
- If properties are same → priority is :
  - 1) Inline
  - 2) Embedded (<style>)
  - 3) External (\*.css file)

Q) If styles are defined using type, class, and id selectors.

- If different properties → All apply
- If properties are same → priority is :
  - 1) Id (#idname)
  - 2) Class (.classname)
  - 3) Type (h1, p, div etc.)

## Bootstrap components

04/09/25

- \* Bootstrap components are pre-styled UI elements that developers can use to quickly build a webpage without everything from scratch
- \* `<script src = ".../node_modules/bootstrap/dist/js/bootstrap.bundle.min.js" ></script>`
- \* Only write the link within the body section.

### 1) Alerts

- \* An Alert is an embedded message box used to display notifications, warnings, success message or errors.

#### Classes:

- 1) `.alert` → Base class for creating an alert.
  - 2) `.alert-[contextually]` → colors.
- Ex:-
- `.alert-primary` (info message) (Blue)
  - `.alert-success` (success message) (Green)
  - `.alert-danger` (error/danger message) (Red)
  - `.alert-warning` (warning message) (Yellow)

#### Syntax:

`<div class = "alert alert-success" > this is a success alert msg </div>`

- 3) `.alert-dismissible` → Makes alert closable (dismissible)

- 4) `.alert-link` → styles link inside alert.

#### Syntax:

`<div class = "alert alert-success alert-dismissible" > </div>`

#### jQuery | JavaScript Attributes:

- \* `data-bs-dismiss = "alert"` → close the alert when button is clicked

Syntax :

```
<button data-bs-dismiss="componentName"></button>
```

componentName → can be alert, modal etc

a) Modals:

- \* Modal is a dialog box that pops up in page
- (\*) \* Modal is a dialog box/ popup that appears on top of the current page content
- \* Hidden by default, shows using Javascript/JQuery attributes.

classes:

a) .modal → Base class

Syntax :

```
<div class="modal"></div>
```

b) .modal-dialog

Defines the modal box position.

c) .modal-dialog-centered

Vertically centres the modal

d) .modal-dialog-scrollable

Adds the scrolling inside the modal body if content is long.

e) .modal-dialog-fullscreen

Expands Modal to full screen (can also use, -sm-down, -md-down).

f) .modal-content

It is used to put the actual content, which contains the header, title, body,

footer

g) .modal-header :

It is used for the title part, close buttons or avatars or image etc

h) .modal-body :

It is used for main content (forms, texts etc)

### 1) modal-footer

It is used for footer buttons (close buttons / save buttons / save changes buttons)

2) .fade → Adds fade-in/out animation effect:

Syntax:

```
<div class = "modal">
  <div class = "modal-dialog">
    <div class = "modal-content">
      <div class = "modal-header">
        </div>
      <div class = "modal-body">
        </div>
      <div class = "modal-footer">
        </div>
        <div>
          </div>
        </div>
        <div>
          </div>
        </div>
```

NOTE:

Modal is a dialog, which is hidden by default, you have to open modal by using jquery attributes.

Attributes:

a) data-bs-toggle = "modal" → tells bootstrap this element should open modal

b) data-bs-target = "#myModal" → points the ID of the modal

### 3) Badges

A badge is a small lightweight component used to display counts, labels, highlights.

- commonly used placed next to buttons, headings, links.

Classes

a) .badge → the small count or notification dot.  
+ Base class of all badges

b) .badge-{contentually} → Background colour  
(EX: bg-warning, bg-primary etc.)

c) .text - {contextual} → Text color  
 ex :- Text-primary, text-success etc

Syntax:

```
<button type="button" class="btn btn-success">Notification
  <span class="badge bg-light text-dark">4</span>
</button>
```

## 4) Cards

\* A card is a flexible and extensible "container".  
 \* It includes options for headers, footers, images, titles, text and more.

classes:

a) .Card → Base class of all cards

b) .card-header  
 Top section (usually for titles, headings, navigations)

c) .card-body  
 Middle section (main content)

d) .card-footer  
 Bottom section (buttons, links, or small text)

e) .card-title  
 Title inside .card-body

f) .card-subtitle  
 Subtitle text inside .card-body

g) .card-text  
 Regular text inside .card-body

h) .card-img-top  
 Image at the top of the card

i) .card-img-bottom  
 Image at the bottom of the card

j) .card-img-overlay  
 placed text on top of an image

### 5) Spinners

\* spinners are lightweight loading indicators that shows the system is processing/loading data.

classes:

a) +spinner-border (Default)

\* creates a border-style rotating spinner  
\* default it is hollow circle.

b) +spinner-border-sm

small version of border spinner

c) +spinner-border-lg

large version of border spinner (Custom sizes usually via CSS)

d) +spinner-grow

creating a growing a dot spinner

e) +spinner-grow-sm

small version of grow spinner

f) +spinner-grow-lg

large version of grow spinner

h) +text-icontextually  
changes the spinner color (primary, secondary, danger, warning, success, etc)

syntax:

```
<div class = "spinner-border" ></div> // For Border spinner
```

```
<div class = "spinner-grow" ></div> // spinner grow
```

### 6) Progress

\* It refers to the status or completion level of a task

\* A progress bar visually shows how much of a task has been completed and how much is left

Ex: File upload (50% done), Download status, Loading process, Form completions

## classes

### a) • progress

\* Base class of all progress.

### b) • progress-bar

\* This is the filled part of the bar that shows how much of the task is done.

### c) • progress-bar-striped

\* Adds diagonal stripes on the progress bar for visual style.  
\* Just makes the bar look more interesting, like a moving pattern.

### d) • progress-bar-animated

\* Animates the stripes so they slide continuously, giving the effect of an active process.  
\* Usually combined with •progress-bar-striped.

### e) bg-color

\* Changes the color of the progress bar (Ex: bg-primary, bg-success etc).

### f) style = "width: X%"

\* Sets how much of the progress bar is filled, representing task completion.

## Syntax:

```
<div class = "progress">
```

```
<div class = "progress-bar progress-bar-striped progress-bar-animated" style = "width: 30%;"> 30% completed.
```

```
</div>
```

```
</div>
```

\* By using Carousel, we can create sliding or scrolling content (images/text/banners etc)

## classes:

### a) • carousel

\* Main container / Base class of all carousels.

### b) • carousel-inner

\* A container inside •carousel

\* Holds all the slides (items)

## (1) .carousel-item

- \* Each slide (image/text) is a .carousel-item
- \* must be placed inside ".carousel-inner"

## (2) .active

- \* Marks which slide is visible first
- \* only one item should have .active

syntax:

```
<div class="carousel">
```

```
  <div class="carousel-inner">
```

```
    </div>
```

```
  </div>
```

- \* carousel will start with any item by default. you have to set any one item as ".active".

syntax:

```
<div class="carousel-item active">
```

- \* carousel will not start animation by default, you have to enable animation by using

the attribute "data-bs-slide";

syntax:

```
<div class="carousel" data-bs-slide="carousel">
```

- \* Carousel Animation requires 2 different animation style, you can any one of both style.

a) slide

b) carousel-fade

syntax:

```
<div class="carousel slide">
```

```
  <div class="carousel-fade">
```

## Carousel controls:

- \* They allow to navigate b/w slides

- \* you can move to previous or next

classes:

a) .carousel-control-prev → previous button ( $\leftarrow$ )

b) .carousel-control-next → next button ( $\rightarrow$ )

### c) .carousel-control-prev-icon:

- \* the icons (arrows) inside the previous button.
- \* Bootstrap gives a default left arrow.

### d) .carousel-control-next-icon:

- \* the icons (arrows) inside the next button.
- \* Bootstrap gives a default right arrow.

### Carousel Control Attributes:

#### a) data-bs-slide

- \* Tells which direction to move the carousel.

values:

- "prev" → move to previous slide
- "next" → move to next slide

#### b) data-bs-target

- \* Tells the button which carousel to control.

values:

ID of the carousel (#).

Syntax:

```
<button class="Carousel-Control-prev" data-bs-slide="prev" data-bs-target="#targetBanner">
```

```
<span class="Carousel-control-prev-icon"></span>
```

```
</button>
```

### Carousel Indicators:

- \* It allows to navigate to any specific slide directly.

- \* small dots at the bottom of the carousel.

- \* they allow users to jump directly to a specific slide.

classes

a) .carousel-indicator → container for all dots.

b) .active → makes the current active dot.

Attributes:

data-bs-slide-to = "indexNumber" → tells which slide to go (start from 0)

- Note:
- You can control the slide timing by using the attribute "data-bs-interval"
- Controlling slide Timings:
- \* use `data-bs-interval = "time-in-ms"` inside `<carousel-item>`
- \* Default = 5000 ms (5 sec)
- Syntax:  
`<div class="carousel-item" data-bs-interval="3000"> </div>`      3000 millisecond = 3 sec.
- g) Navbar
  - \* A Navigation bar is used to create menus (Home, About, Contact etc.)
  - \* It is responsive → automatically collapses into a hamburger menu on small screen.
- classes:
  - a) `.navbar`
  - \* Main container for the navbar
  - b) `.navbar-expand-lg | md | sm`
  - \* Controls when navbar collapses.
    - Ex: `.navbar-expand-lg` → expands on large screen and up
    - `.navbar-expand-sm` → expands on small screen and up
  - c) `.navbar-toggler`
  - \* the hamburger button (≡) expanding menu
  - d) `.navbar-brand`
  - \* used for logo or brand name
  - e) `.navbar-nav`
  - \* Container for the nav menu links
  - f) `.nav-item`
  - \* Each menu item (Home, about, contact)
  - g) `.nav-link`
  - \* the actual clickable link inside `.nav-item`
  - h) `.navbar-collapse` → the collapsible area (wraps nav items)
  - i) `.navbar-dark` and `.navbar-light`
  - \* Controls text color of navbar

### iii) bg-\* or bg-fcontentual}

\* sets background colors for Navbars

\* → primary, secondary etc.

### iv) Collapse

\* Dynamically show or hide any element on the page

#### classes

- a) .collapse → applies collapse behaviour to an element
- b) .show → makes the collapsed element visible

#### Attributes:

- a) data-bs-toggle = "collapse"  
specifies that clicking the element will toggle collapse behaviour.
- b) data-bs-target = "#id"  
points to the elements you want to collapse/expand

### v) Accordion

\* A group of collapsible items only one item can be open at a time (mutual exclusion).

#### classes:

- a) .accordion → wrapper for the entire accordion
- b) .accordion-item → Each individual item in the accordion
- c) .accordion-header → header section that contains the button
- d) .accordion-button → clickable button that toggles the collapse
- e) .accordion-collapse → collapsible section for content
- f) .accordion-body → main content inside the collapsible section
- g) .accordion-footer → options/footer for each item

#### attribute:

data-bs-parent = "#accordionId" → Ensures only one item is open at a time

### vi) Dropdown

\* Displays a list of options when triggered

\* including images, icons, symbols.

## Classes.

- a) .dropdown → the main wrapper for the dropdown
- b) .dropup → make the dropdown open upward
- c) .dropstart → opens to the left
- d) .dropend → opens to the right
- e) .dropdown-toggle → makes the button or link trigger the dropdown menu
- f) .dropdown-menu → container of all dropdown items
- g) .dropdown-item → Each clickable option inside the menu
- h) .dropdown-divider → adds the horizontal separator b/w items
- i) .dropdown-menu-dark → dark-themed dropdown menu

## JAVASCRIPT

27/9/25  
saturday,

- \* JAVASCRIPT (JS) is a programming language that developers create interactive web pages.
- \* Javascript is a lightweight, Object Oriented programming language.
- \* Javascript is a JIT-compiled and Interpreted programming language.
  - Interpreter → translates code line by line
  - Compiler → translates all code at once.
- \* It is a single-threaded language that executes one task at a time.

\* Javascript extension is ".js" or "Js"

\* Current Javascript Version ~~ES6~~ ~~ES7~~ ~~ES8~~ ~~ES9~~ ~~ES10~~ ~~ES11~~ ~~ES12~~ ~~ES13~~ ~~ES14~~ ~~ES15~~ ~~ES16~~ ~~ES17~~ ~~ES18~~ ~~ES19~~ ~~ES20~~ ~~ES21~~ ~~ES22~~ ~~ES23~~ ~~ES24~~ ~~ES25~~ ECMAScript 2025

### Compilation Techniques

#### 1) JIT (Just In Time)

- code is compiled inside the browser while executing
- useful for client side scripting

#### 2) AOT (Ahead of Time)

- code is compiled ~~inside the browser~~ before execution at application level
- faster and improves application performance.

### Features of Javascript

- |                   |                                      |
|-------------------|--------------------------------------|
| 1) Lightweight    | 3) Object-Oriented Programming (OOP) |
| 2) Dynamic Typing | 4) Procedure-Oriented                |
|                   | 5) Interpreted Language              |

### Usage of Javascript

Client-side — HTML, React, Angular, Vue

Server-side — Node.js

Database — MongoDB

Animations/multimedia — Flash, Actionscript

## Role of Javascript on Client-side

→ Reduce server load and improve user experience

### a) DOM Manipulations

- Add elements to DOM

- Remove if from it

- Update Content in elements

- Dynamically configure styles

↳ validations

- Form validations before sending data to server

### b) Browser Interactions

- Detect browser versions & compatibility

- prevent unnecessary server requests.

- Manage plugins & extensions

- Access client location & history

- Perform browser window operations

## Evolution of Javascript:

1) ECMAScript was introduced for web scripting in 1990.

2) 1995: Netscape developed the Netscape Navigator browser.

- Brendan Eich created a scripting language called "Mocha"; later renamed

"LiveScript".

3) Netscape handed LiveScript to Sun Microsystems, which renamed it "JavaScript".

4) 2002: Netscape stopped development and transformed JavaScript standardization to ECMA International.

## versions of Javascript

ES5 - 2015

ES6 - 2016

ES7 - 2017

⋮ ⋮

ES22 → 2022 (current version)

## Integrating Javascript into HTML pages:

### 1) Inline javascript

- javascript is written directly inside HTML elements
- fast to implement
- cannot reuse functions easily

Syntax:

```
<button onclick="window.print()">print</button>
```

### 2) Embedded Javascript

- javascript is written inside a <script> tag in HTML page
- can be placed in <head> or <body>

Syntax:

```
<script type="text/javascript">
    function printpage()
    {
        window.print();
    }
</script>
```

```
<button onclick="printpage()> print</button>
```

MIME Type: 'text/javascript' or 'language="javascript"

{ "use strict"  
 } It used for make your Javascript  
 safer and cleaner, enforcing stricter  
 rules, catching common mistakes  
 and preventing some silent errors

### 3) External Javascript:

- javascript functions are placed in a separate .js file
- allows reusability across multiple pages

Ex: 1) Create file: print.js

```
function printpage()
{
    window.print();
}
```

2) Link in HTML

```
<script src=".\\src\\print.js" type="text/javascript"></script>
```

Q: How javascript refers HTML elements?

- a) i) By using DOM Hierarchy (Index Number)
- ii) By using Reference Name
- iii) By using Unique ID
- iv) By using CSS Selector

i) DOM Hierarchy → HTML as objects + Tree structure + javascript can control it

\* DOM → Domain Object Model → It is an interface

\* DOM hierarchy is a tree-like structure that represents an HTML or XML document.

\* DOM is the faster technique implemented for rendering dynamic elements.

\* however changing the position of element in page requires the change in index number every time.

Syntax:

window.document.form[0].element[0]

\* By using DOM,

- a) Create New Elements
- b) Add Elements
- c) Remove Elements
- d) Delete Elements
- e) Change styles (CSS)
- f) Add Event Listeners
- g) Change Attributes

Methods to select HTML elements:

i) document.getElementById('idname') (Recommended) ✓

\* Returns the element with specific id

\* Select one element

\* Needs Unique Id

## No Index Based

Syntax:

```
document.getElementById("id");
```

Ex:-

```
document.getElementById("title").style.color = "Red";
```

### 1) getElementByTagName()

Returns list of all Elements with the specified tag.

Index-Based C Returns HTML Collection.

Selects all tags of same type.

Affects others by the position change

Syntax:

```
document.getElementsByTagName("tagName") [indexNumber];
```

```
Ex: document.getElementsByTagName("img") [0].src = "photo.jpg";
```

### 2) getElementByClassName()

Returns list of all elements belongs to the specified class.

\* Returns list of all elements Required

\* Index-Based Required

\* Selects all elements with same class.

Syntax:

```
document.getElementsByClassName("className") [indexNumber];
```

```
Ex: document.getElementsByClassName("box") [1].style.backgroundColor = "Green";
```

### 3) getElementsByName()

\* Mostly used in forms

\* Returns NodeList → index Based

Syntax

```
document.getElementsByName("name") [index];
```

```
Ex: document.getElementsByName("username") [0].value = "thiru";
```

### 5) Query Selector () (Recommended)

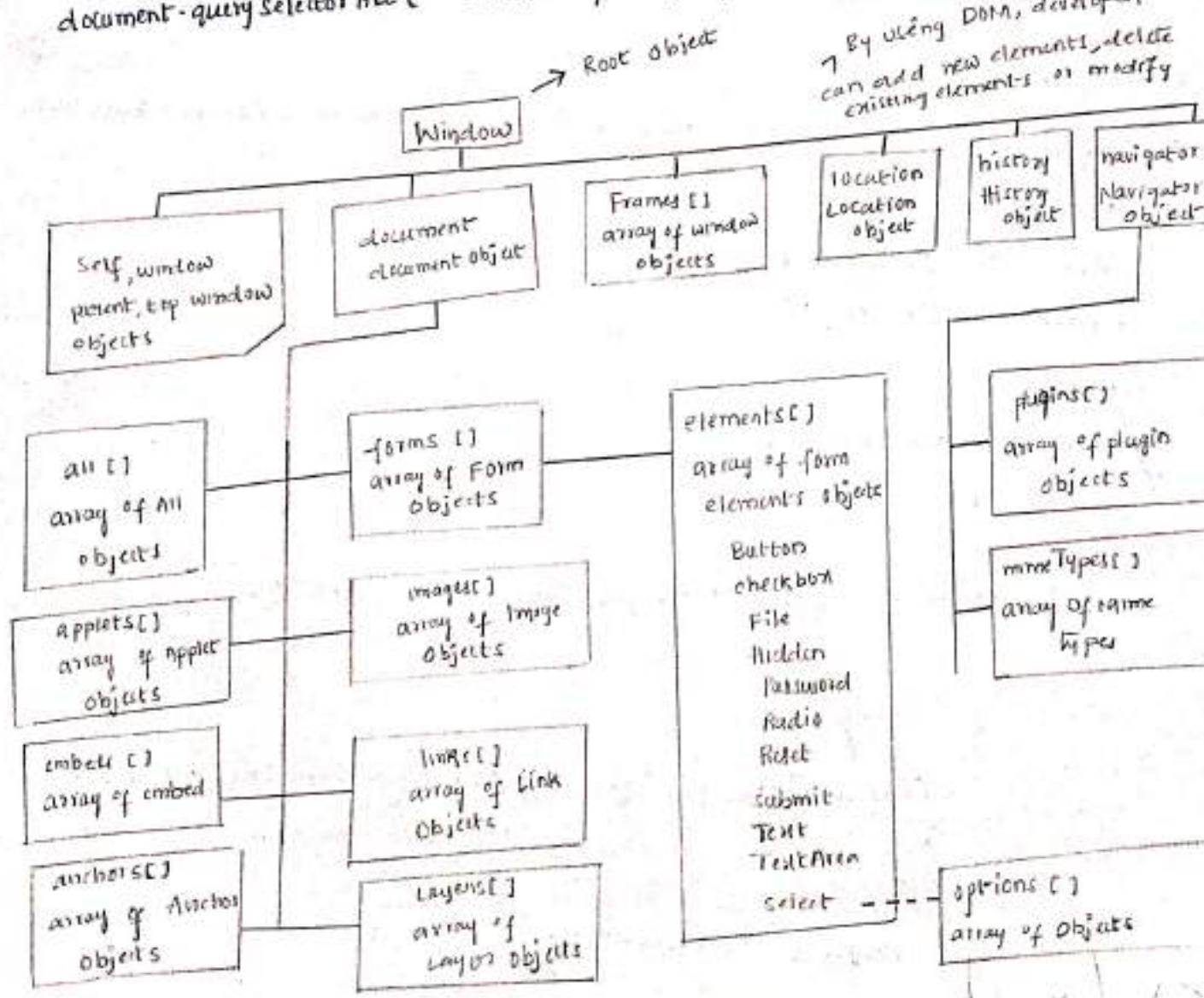
- \* Selecting the first matching element
- \* Returns only one element
- \* Returns the first object matching CSS style selector

Syntax:  
`document.querySelector (" .className / #id / tagName " )`

### 6) Query Selector All () (Recommended)

- \* Selecting all matching elements
- \* Returns a NodeList (array-like)
- \* Returns all objects matches the CSS style selector
- \* You must use an index number

Syntax:  
`document.querySelectorAll (" .className / #id / tagName " )`



## Javascript Output Techniques

### 1) alert()

\* Shows a popup message box

\* Used to display plain text only

\* It cannot have formatted text (HTML tags, Images, CSS styling).

\* User must click OK to continue and NO option to cancel

Syntax:

```
alert("your message")
```

```
alert("Line 1 in Line2");
```

```
alert(10+20); (or) alert(Expression);
```

Ex:

```
alert("Form submitted successfully!");
```

### 2) confirm()

\* Confirm() is similar to alert() but has OK and Cancel buttons

\* Returns a boolean

- true → OK clicked

- False → Cancel clicked

Syntax:

```
confirm("Are you sure?");
```

Ex:

```
let result = confirm("Do you want to delete this?");
```

```
if (result) {
```

```
    confirm("Deleted..."); // When you click OK, this will print,
```

```
}
```

```
else {
```

```
    confirm("Cancelled"); }
```

### 3) console()

- \* Console is a developer CLI (Command Line Interface)
- \* It is developer provided with browser developer tools (inspect).
- \* Console = Browser developer tools (inspect → console tab).
- \* Developer can log all messages as output in console.
- \* Only for Debugging, which will see the how internally it works.

#### Common methods

- a) console.log() → Normal messages
- b) console.error() → error messages (Red color)
- c) console.warn() → Warnings (Yellow color)
- d) console.info() → info messages
- e) console.debug() → debugging messages.

#### Syntax:

```
console.log ("string | expression");
```

- \* Console cannot take Markup or Complex representation
- \* Console methods are only for designs not for production.

### 4) a) innerHTML

- \* innerHTML is used to get or set the HTML content inside an element.
- \* By using innerHTML, we can also the Markup (HTML tags) using javascript.
- \* It allows you to change content of webpage dynamically.
- \* It can include HTML tags, not just Text.

#### Syntax:

```
Object.innerHTML
```

Object.innerHTML = value → It represents the content of the html element

↓ To set the innerHTML property

Ex:

```
<div id="msg"> Hello ! </div>
<button onclick="changeText()">click me </button>

<script>
function changeText() {
    document.getElementById("msg").innerHTML = "<b> Welcome to javascript </b>";
}
→ When button click content changes from Hello to welcome to javascript (in bold)
```

5) OuterHTML → changes entire element including the tag

- \* OuterHTML is similar to innerHTML, but it will replace the existing markup with new markup
- \* Replace the whole tag or Replace the whole element

Syntax:

```
element.outerHTML = "HTML-structure";
```

Ex: <div id="box"> Hello </div>

```
document.getElementById("box").outerHTML = "<p> Kutra Thirumalai </p>";
```

Output

```
<p> Kutra Thirumalai </p>
```

| outerHTML → to remove whole div tag and  
| new <p> tag appears instead.  
| → original tag does not exist anymore

Difference b/w innerHTML and OuterHTML

innerHTML

Returns only the inside the content  
of an element

Replace the only the content  
inside the element

Ex: update text, image,

OuterHTML

Returns the entire element  
including the tag itself

Replace the element itself  
(whole tag + content)

completely replace element  
with new HTML

## 6) innerText

- \* innerText is a DOM property used to get or set or change the visible text inside an HTML element.
- \* It ignores HTML tags and it is a RC datatype
- \* It does not include hidden text (css: display: none, visibility: hidden)
- \* It will not allow formatted text
- \* what is visible after CSS rules are applied.
- ?? document.write()
- + It is used to print text/ HTML directly on the webpage, but only while the page is still loading.

Syntax:

```
document.write("msg" | markup | expression);
```

## 8) textContent

- \* textContent is a DOM property used to get or set the plain text inside the element.
- \* Returns Only text
- \* Ignores any HTML tags.
- \* Faster than innerText
- \* Treat everything as plain text
- \* All text, without considering CSS.

Syntax:

```
let text = element.textContent; // To retrieve the text content of an element
element.textContent = 'NewText'; // To set or modify text content of element
```

Ex:

```
<div id="demo">Hello <b>World</b></p>
document.getElementById("demo").textContent; // textContent = "<b>Hello</b>"
```

Output:

Hello world

Output  
-110

## Java Input Techniques:

- 1) `prompt()` — Input from browser
  - \* `prompt()` method provides a popup that allows "input" from the Browser
    - (a) shows a popup box asking the user to type something from the browser
  - \* Empty string [ ' ' ] : Ok click without value
  - \* String Value : Ok click with value
  - \* null : cancel clicks with or without value

### Syntax:

```
n prompt ("your message");
```

Ex: `prompt ("Enter Something");`

```
n prompt ("your message", "default text");
```

```
n prompt ("Enter your name:", "thim");
```

```
alert ("Welcome" + name);
```

### 2) Form input elements

- \* You can design the form using various "input elements" such as textbox, password, number, email, url, radio, checkbox, dropdown.
- \* The above elements allow users to enter different types of data.
- \* Every form input stores the user-entered data inside its "value" property.

Ex:

```
<input type = "text" id = "name" value = "John">
```

```
document.getElementById("name").value.
```

- 3) Query String
  - \* Query string is the part of the URL that carries that extra information from one page to another.
  - \* It starts with a ? and contains key = value pairs.
  - \* Key must be unique and value may be not unique
  - Ex: `https://amazon.in?category=mobiles`
  - \* You can access query string in page by using javascript "location.search".
  - \* "location.search" Returns complete Query strings. you have to string methods extracts only value.
  - `str = location.search;`
  - `str.indexOf(indexNumber)` → Reads from specified index upto end
  - `str.substring(str.indexOf("="), str.indexOf("=", indexNumber + 1))` ; from q upto end
  - `str.substring(str.indexOf("=") + 1)` → from 10 to end.

### Javascript Basics:

- 1) Variables:
  - Variables are storage locations in memory where you can store a value and use it as a part any expression.
  - declaration
    - `let x;`
    - `x = 10; ✓`
  - Assignment
    - `'use strict'`
    - `x = 10; x`
  - initialisation
    - `let x = 10;`
    - `'use strict'`
    - `var x = 10; ✓`

Note Javascript allows to do directly assignment without declaring if it is not in strict mode

\* If Javascript is in strict mode declaring variable is mandatory.

Javascript variable by using 3 keywords.

### a) Var

\* It defines a function scope variable  
\* you can declare in any block of a function and can access from any another block inside the function

\* It allows declaring, assignment, initialization

\* It allows shadowing process of re-declaring or re-initializing with same ~~to~~ name identifier within the scope.

Ex: function f()

{

var x; → declaring

Var x=10; → initializing

if (x==10)

{

var y = 20;

Var y=30; → shadowing

}

document.write('x' + x + '  
y' + y);

}

\* It allows hoisting:

↓  
process of the declaring and a using a variable anywhere.

there is no order dependency. you can use and declare later in functions

### Syntax:

use strict"

x=10;

document.write('t' + x);

var x; // hoisting

- b) let → can change
  - \* It defines the block scope variable
  - \* It is accessible within the specified block and to its inner blocks  
[this mechanism is called as closure]
  - \* It supports declaring, assignment, initialization.
  - \* It will not support shadowing, hoisting
- c) const → It is a variable value cannot be changed once it is assigned
  - \* It defines block scope variable
  - \* It will support only initialization
  - \* It will not support declaring and assigning.
  - \* It will not support shadowing, hoisting.

### Variable Global Scope

- Global scope defines the accessibility of variable across the multiple functions.
  - Declaring or initializing variable outside function makes as global
  - You can configure a global variable inside the function using "window" object.
- Syntax:
- ```
window.variableName = value;
```

Ex:

```

var x=10;
let y=20;
const z=30;
function f1() {
  window.a=50;
  document.write("x=" + x + "y=" + y + "z=" + z + "a=" + a);
}
function f2() {
  document.write("x=" + x + "y=" + y + "z=" + z + "a=" + a);
}

f1();
f2();
  
```

when we write window object here a value can accessed from both functions f1() and f2() also.

when we cannot write object only we can only f1()

## Variable Naming

- \* Name must start with an alphabet
- \* It can be alpha numeric with '\_';
- \* You can start name with '\_'. But it is not recommended.
- \* ECMAS does not recommend to use special char.
- \* Name max length can upto 255 char.
- \* Never use keywords as variables

Ex:-  
 var for;  
 var if;  
 var while;

- \* Variable name must speaks what it is.

Ex:- Employee employee = new Employee();

- \* Always use Camel Case Ex:- var UserName;

## Javascript Data Types

- \* In computer programming data type defines the data structure.
- \* It is about the type and behaviour of data in memory.
- \* Javascript is not a strongly typed language.
- \* Hence, there is no datatype restriction for data in memory.
- \* var x = any type of data can be stored.
- \* Javascript is implicitly typed language. The data type changes according to value type.
- javascript datatype in memory can be
- ▷ primitive datatype
- \* primitive datatype are immutable type  
 ↓  
 you cannot change the structure
- \* It has a fixed Range for value.
- \* They are stored in memory stack
- \* stack uses LIFO (Last IN First Out)

- Javascript primitive types are

- a) Number      b) string      c) Boolean      d) null      e) undefined      f) BigInt
- g) symbol

a) Number type

A number for Javascript can be any one of the following.

signed integer      -5

unsigned integer      5

float point      34.32 (2 places)

double      345.32 (3 places)

decimal      2345.32 (29 decimal places)

exponent       $2 \times 10^3$  [ $2 \times 10^{13}$ ] = 2000

hexa      000fx45

octa      00578

binary      0b1010

f) BigInt

BigInt is a new datatype for safely using large integers.

BigInt configures large integer values with "n" as suffix.

Syntax: var x = 99889776543; // It is not safe.

var x = 99889776543n; // It is safe

b) String

\* String is a sequence of characters used to represent the text

\* String can be created with single quotes ('...'), double quotes ("...") or

backticks (`...`)

Ex: let name = "thru"; or let name = new String("thru");

\* Strings are immutable (cannot be changed directly)

\* String have length

\* String supports many methods (Replace, concat, toUpperCase etc)

\* Each character in a string is assigned in index and starts from 0

\* You have to parse [convert] string into Number

a) parseInt — converting string into integer

b) parseFloat — converting string into decimal (or float)

Syntax:

```
Var age = parseInt(prompt("Enter Age"));
document.write("you will be " + (age + 1) + " next year");
```

\* Some imp methods : substring, toUpperCase, toLowerCase, concatination, Replace, length, indexOf etc.

\* You can verify the input type number or not by using Javascript method " isNaN()".

FAQ - How to verify the input value is number or not using Regular Expression

- By using pattern and javascript "match()

Syntax:

```
Var age = '30';
Var regExp = /\d{2}/;
If (age.match(RegExp))
{
}
```

\* You to convert a number into string.

a) toString() - Converts a value (number, boolean, array,) into a string.

b) toLocaleString() - Converts the a number or date into a string based on the user's local settings (country format).

### Java script String Type

1) String is a literal with group of characters enclosed in

a) double quotes " " } used to toggle outer and inner string

b) single quotes ' ' } symbol:

```
Var link = <a href='home.html'>Home</a>;
```

c) back ticks ` ` }

```
Var link = `<a href='home.html'>Home</a>`;
```

\* Binding versions of javascript requires a lot of concat techniques

Using "+" operator

Syntax:

"string" + dynamicValue + "string"

ES5 versions of javascript introduced a data binding expression "`{} {} {}`".

which you can embed in a string with backticks only.

Syntax:

'string {{dynamicValue}} string'

\* string cannot print few characters, to print the non-printable

characters you have to use escape sequence character "\\".

Syntax:

`let path = "d:\\images";`

\* Javascript provides the various methods

\* Javascript string formatting includes

a) bold()

b) italic()

c) fontsize()

d) fontcolor()

e) sup()

f) sub()

g) strike()

h) toUpperCase()

i) toLowerCase() etc

Syntax:

`Var str = "Kumar Thimalesh";`

`document.write(str.bold());`

`document.write(str.bold().italic().fontSize(4).fontColor('green'));`

presentation (o) building a string with a specific style ↓

modification

existing (o)

change string → produce a new string

(o) change text content

\* javascript - strings manipulation :

a) toUpperCase()

b) toLowerCase()

c) length()

d) slice()

e) substr()

f) replace()

g) trim()

h) split()

i) concat() etc..

Events

- ④ **Onblur** → Onblur triggers when user leaves the input field  
 (or) specifies actions to perform when element lost focus  
 Example flow: user click inside text-box → type something → click outside  
 moves another field → onblur runs.

- ⑤ **Onkeyup** → onkeyup triggers every time the user releases a key  
 (or) specifies actions to perform when a key is released.  
 Example flow: press key → key down → onkeyup runs.

Note:

You cannot apply font and style to BC data Element using string methods.

Syntax:  
`document.getElementById('textBox-Id').value = "John:bold()"; // invalid`

→ Javascript provides various methods for string manipulations

→ **length [property]**: It is used return the total count of characters in string

↳ `length` [property] : It is used return the total count of characters in string

Syntax:  
`var str = "Thirumalesh";  
 str.length; // 12`

② **charAt()** : It Returns the character at specified index in string  
 Index starts with "0"

Syntax:

`var str = "Welcome";`

`str.charAt(0); // W`

③ **charCodeAt()** : It Returns the ASCII code of the character at specified index.

`"A" = 65, "Z" = 90`

`"a" = 97, "z" = 122`

Syntax:

```
var name = "Ajay";
str.charCodeAt(0); // 65
str.charAt(0); // A
```

- 4) startsWith(): It Returns boolean true if string starts with specified char(s)
- 5) endsWith(): It Returns boolean true if string ends with specified char(s)

Syntax:

```
Var str = "445567788";
str.startsWith("445"); // true
str.endsWith('445'); // false
str.endsWith('88'); // true
```

- 6) indexOf(): It Returns the index number of character in specified string.
- If character not found then it returns "-1".
- It Returns the first occurrence value index.

Syntax:

```
Var str = "Welcome";
str.indexOf('e'); // 1
str.indexOf('a'); // -1
```

Syntax

String.indexOf(searchValue, startPosition);

- 7) lastIndexOf(): It Returns the last occurrence index number of given value.

Syntax:

```
Var str = "Welcome";
str.lastIndexOf('e'); // 6
```

- 8) slice(): It is used to extract chars between specified index.

- 9) substr(): It is used extract number of chars from given base index.

- 10) substring(): It can extract chars between specified index  
bidirectional.

FAG)

What is the difference between slice(), substr(), substring()?

\* slice() Refers to start index and end index, the end index must be greater than start index

Syntax:

`str.slice(startIndex, endIndex);`

`str.slice(7, 14);`  $\Rightarrow$  7 to 14 index

`str.slice(7);`  $\Rightarrow$  7 to end

`str.slice(7, 0);`  $\Rightarrow$  invalid

\* Cut from start index to end index (end not included)

\* Allows negative numbers.

\* It is uni-directional

Ex:- J a V a S C r I p t  
 0 1 2 3 4 5 6 7 8 9  
 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1

Ex: "Javascript".slice(-6); // script

Ex: "Javascript".slice(-6, -1); script

starts with s

ends before t because (-1) (end not included)

\* Substr() Refers to the number of characters from the specified index

\* Does not allow negative values

length  
or

Syntax: str.substr(startIndex, howManyCharacters);

Ex:- str.substr(7, 4); // from 7, it can read 4 characters.

str.substr(7); // from to end.

str.substr(7, 0); // empty

\* substring() Refers to the start and end index bi-directional

Syntax:

`str.substring(startIndex, endIndex);`

`str.substring(7, 14);` // 7 to 14 in str

`str.substring(7);` // 7 to end

`str.substring(7, 0);` // 7 to start

\* -ve values are not allowed may be allowed. Javascript treats them as 0.

### Rules for Substring

1) No negative numbers allowed

"Thirumalesh".substring(-3, 4);

// it treats as substring(0, 4); // thru

2) if startIndex > endIndex → Javascript swaps them automatically

"Thiru".substring(6, 2);

(b) substring(2, 6);

11) `split()`: It can split the string into an array by using specific delimiter (or) separator

Syntax: `string.split(separator, limit)`

Var details = "John-23, david-25, sam-31";

details.split(","); // [John-23, david-25, sam-31]

"One two three".split(" ", 2); // ["One", "two"]

"Hello World" → ["Hello", "World"]

1) " " split by space

(or)  
Break sentence into words

"a, b, c" → ["a", "b", "c"]

2) ", "

split by comma

"2024-10-21" → ["2024", "10", "21"]

3) "-"

split by dash

"Hi" → ["H", "i"]

4) ""

split every character

12) `trim()`: It is used to Remove leading spaces from a string and return string without leading spaces

→ Space between before and after

\* does not Remove → Space between words.

Syntax:

`string.trim();`

Ex: Var str = " Welcome "

`str.trim();` // Welcome

" Welcome java ".trim();

// welcome java

12) `match()`: It is used to compare your string with a regular expression and return true if string is according to the given expression.

Note:

In javascript Regular Expression is written in " / / ".

Syntax:

`Var regExpression = /\d{10}/; // Exactly 10 digits number.`

`Var str = "999999";`

`if (str.match(regExpression))`

{

  true;

}

else {

  false;

}

13) `concat()`: It is used to join two or more strings and returns a new string.

Syntax:

`string.concat(string1, string2 ...);`

`Ex: "Hello".concat(" ", "world"); // Hello world`

`Ex: "Hello".concat(" ", "world"); // Hello world`

14) `replace()`: Returns only the first matching substring with another value.

Syntax:

`string.replace(searchValue, newValue);`

`Ex: msg = "javascript is a language";`

`msg.replace("javascript", "Java"); // Java is a language`

`msg.replace("javascript", "Java"); // Java is a language`

15) `repeat()`: Repeat the string multiple times.

`Ex: "hi".repeat(2); // hi hi`

Syntax:

`string.repeat(count);`

16) `includes()`: Check if a string contains a given substring.

Syntax:

`string.includes(searchValue);`

`Ex: "I am thru".includes("am"); // True`

17) `padStart()` / `padEnd()`: Add characters to the beginning of a string until it reaches given length.

`string.padStart(targetLength, padding)`

`Ex: "7".padStart(4, "0"); // 0007`

**Boolean**

- Boolean are used in decision making.
- Boolean types in javascript can handle the keywords "true" and "false".
- Boolean conditions in javascript can manage using "1" and "0".

1 = true

0 = false

$$\text{true} + \text{true} = 2$$

$$\text{true} + '1' = \text{true}$$

$$\text{true} + 10 = 11$$

$$\text{true} - \text{true} = 0$$

**Undefined**

value

- \* It Refers to a memory where is not defined.
- \* "Undefined" Keyword to verify the value in memory.

**\*\***  
In javascript, the default value is "undefined".

FAQ) What is the difference between undefined and not-defined?

undefined → value is not defined (or) value is not there

not-defined → the memory reference is not-defined. (or) Reference is not there

x = undefined

y = not-defined.

Ex: `Var x;`  
`document.write('x=' + x) <br> y=' + y);`

**Null**

Null is a type configured memory reference when value is not providing during the Runtime.

Ex: `let a = null;``console.log(a); // null.`

\* It is verified by using "null" keyword.

**Symbol**

- \* It is configured a hidden key for object.
- \* It is configured a hidden key for iterations. But they are accessible individually.
- \* Hidden key are ignored on iterations. But they are accessible from version ES6.
- \* Symbol is a new datatype of a javascript for developers from version ES6.

**Syntax:**`Var key = Symbol();``Var obj = {``[key]: value }``* name : 'thru';``}`

`obj[key] ⇒ can access the value stored in key: Reference`

## Non-primitive Data types

- They are mutable types
- They don't have a fixed range and structure.
- It changes according to memory available
- It uses heap memory. [heap will not have any order of access]
- Javascript non-primitive types are

### a) Array

#### b) Object

#### c) Map

- starts from 0 up to the  $n-1$*
- a) Array → Array is a contiguous collection of values in a sequential order, which you can access Randomly - introduced into computer programming to Reduce overhead and Complexity.
- \* Arrays can Reduce overhead by storing values in sequential order.
- \* Arrays can Reduce complexity by storing the multiple values under the Reference of single Name.
- \* Arrays can store various types of values.

- \* Array size can change dynamically.
- \* Array index starts from 0 to  $n-1$ .

Note: Some of the software programming technologies cannot allocate different types of memory in sequential order and cannot change size dynamically.

[Ex: Java cannot change dynamically] and Java then will use Collections.  
Hence they restrict array if for similar types of values.

Javascript array can store various types and can change the size dynamically.

- \* Array refers to a type of arrangement, where elements are arranged sequentially but can be accessed in Random.

Ex: Keyboard is having an array of keys.

## Declaring Array:

- \* Javascript array is declared by using var, let, const keywords.
- \* It uses a reference name for storing values.

### Syntax:

1) let arrayName;

2) Using array literal (most common)  
let arrayName = [];

### (3) Using Array Constructor

let arrayName = new Array();

### Assign (or) Initialise the Array

- a) [] → static memory [the memory allocated for first reference will continue across multiple requests]
- b) Array() → dynamic memory [which is a newly allocated everytime it's requested]

### Syntax: Initialization

Var arrayName = [];

Var arrayName = new Array();

### Assignment:

Var arrayName;

arrayName = [];

(or)

arrayName = new Array();

Ex: let name = ["thiru", "shiva", "baskar"];

let colors = [];

colors[0] = "Red";

colors[1] = "Green";

let num = new Array(1, 2, 3, 4, 5, 6); (or) let num = new Array(4)

Creates an empty array with length 4, not elements

- \* You can store the any type of values in array memory Reference

a) primitive

Ex 1): Var values = [];

b) non-primitive

values[0] = 1; values[1] = "thiru";

c) Function

values[2] = "true"; values[3] = ["dehi", "hyd"];  
values[4] = function () { document.write("Hello array"); }

- \* We can access array elements [values] by Using Index Number.

Ex: document.write(values[1]); // thiru → Index starts from 0

```
document.write(values[3][0]); // delhi
```

```
document.write(values[4][0]); // Hello Array
```

[4][0] → IIFE (Immediately Invoking Function Expression)  
 ⇒ Function without name is called Anonymous Function

Ex 2 : values =  $\begin{bmatrix} [0] & [1] & [0] & [1] \\ [10, 20, 30], [40, 50, 60] & [[A, B], [C, D]] \\ [0][1][2] & [0][1][0][1] \\ [0] & [0][1] \end{bmatrix}$

```
document.write(values[1][1][0]); // C
```

Array properties and Methods :

- Array properties and Methods :  
 1) finding the length of array, which is the total count of elements in array.

Ex:- values = [10, 20, 30];  
 values.length; // 3

2) Reading the values from Array

a) toString() : It Reads all elements and converts into a string separated with ", ".

b) join() : It is similar to string but uses custom delimiter.

c) slice() : It Reads elements between specified index.

d) map() : It is an iterator that returns all elements using a specific presentation.

e) forEach() : It is similar to map() but can return index of every element in iteration

f) find() : It Returns only the first occurrence elements that matches given condition

g) filter() : It Returns all elements that match the given condition

h) reduce() : It performs specific operation of values returned by iterator

Syntax:

collection.toString() // A, B, C

collection.join(",") // A → B → C

collection.slice(1, 2) → B

collection.map(function (value) {

})

collection.forEach(function (value, index) {

)

collection.find(function (value) {

return condition-for-value;

)

Ex:- var sales = [45000, 42100, 67000, 68000, 73000];

sales.find(function (value) {

return value > 60000; || 67000

});

collection.filter(function (value) {

return condition-for-value;

});

Note:

You can also use various statements explicitly to Read values from array  
for, while, do while, for-in, for..of

i) Explicitly Iterators for Array:

a) for..in : It Reads all properties (or) Indices of Array.

Syntax:

for (var property in collection)

{

}

b) for..of : It Reads all values of Array

Ex:-

var sales = [45000, 42100, 67000, 68000,  
73000]

for (var property in sales) {  
document.write(property); }

// 0

// 1

// 2

// 3

// 4

// 5

Syntax:

for (var value of collection)

{

}

Note: Destructuring Array \*\*\* ✓

JavaScript ES6 Version introduced Array Destructuring

Syntax:

`Var collection = [10, 20, 30];`

`Var [a, b, c] = collection;`

`a = 10`

`b = 20`

`c = 30`

Ex: `Var a = [1, "TV", ["Delhi", "Hyd"]];`

`Var [id, name, cities] = a;`

`id: {{id}} <br>`

`name: {{name}} <br>`

`cities: {{cities}} <br>`

Output

`id = 1`

`name: TV`

`cities: Delhi Hyd`

②

Dynamically Creating HTML Elements:

1) HTML Element can be created in Javascript by using DOM Method.

Syntax:

`document.createElement("tagName")` => Node Name for Element  
 Paragraph => P  
 img => img

2) Assign a memory reference for newly created Element

`Var pic = document.createElement("img");`

3) Configure the properties for element [you cannot use attributes]

`pic.src = "images/name.jpg";`

`pic.height = "300";`

`pic.className = "css className";`

4) Add Element into page : You can add any dynamic element using the DOM method

Syntax:

`parent.appendChild();`

Ex: `<figure> </figure>`

`document.getElementById('figure').appendChild(pic);`

## Presenting Array Elements in HTML Page

- 1) Use any iterator for Reaching elements from Array.  
maps(), forEach(), find(), filter(), for...of, for...in
- 2) In every Iteration creates a new element using DOM Method.
- 3) In every Iteration set properties for new Element.
- 4) In every Iteration append the child element to its parent.

Ex:

```
Var values = [10, 20, 30];
```

```
values.map(function (value) {
  var li = document.createElement ('li');
  li.innerHTML = value;
  document.querySelector ('#ol').appendChild (li);
});
```

### Adding Elements into Array:

- 1) push() : It adds the new Element(s) into array as last Element(s).
- 2) unshift() : It adds new Element(s) into array as first Element(s).
- 3) splice() : It adds new Element(s) into array at specific position.

### Syntax:

```
Var menu = ["Electronics"];
menu.push ("Footwear", ...); // [ "electronics", "Footwear" ];
menu.unshift ("All", ...); // [ "All", "Electronics", "Footwear" ];
menu.splice (1, 0, "kids", ...); // [ "All", "kids", "Electronics",
                                "Footwear" ],
```

### Syntax:

```
splice (startIndex, deleteCount, items ...);
```

### Removing Element into Array:

- 1) pop() : It removes the last Element(s).
- 2) shift() : It removes and Returns the first Element(s).
- 3) splice() : It removes and Returns specific Elements.

```
Ex: var a = ["All", "Electronics", "Fashion"];
      a.pop(); // ["All", "Electronics"]
      a.shift(); // ["Electronics", "Fashion"]
      a.splice(1, 1); // ["All", "Fashion"];
```

### Sorting Array Elements:

- 1) sort(): It arranges elements in ascending order
- 2) reverse(): It arranges elements in Reverse order

Object Type → Object is a data structure that store information in the form of Key = Value

- \* Object is a variable which can hold multiple values.
- \* It is the location where the collection values is stored.
- \* Object is a key and value collection.
- \* Key is "String" and value can be any type.
- \* Values are stored with Reference of keys.

*This → used inside the object  
object name → used outside the object*

### Syntax:

```
Var Obj = {
```

- 'Key1': 'Value1'

- 'Key2': 'Value2'

- "Key3": function() {}

- || more key-value pairs ...

```
}
```

Ex: Var person = {

name: "Thiru",

age: 20,

city: "Kurnool",

}

\* If Object comprises of only data then it is referred as "JSON".

\* JSON - Javascript Object Notation.

\* You can access any value within Object by using "this.keyName".

\* \* "this" is a keyword that refers to current Object

\* You can access the values of object with Reference of "ObjectName.KeyName".

Ex: `document.write(person.name);` // thru

Note: "dot" is ~~ans~~ member invoking operator.

\* JSON data kept is kept in separate JSON file that have the extension ".json".

\* to access data from JSON file Javascript provides a "fetch()" promise.

\* promise is a special type of function that handle the "async" methods.

\* "async" is unblocking technique; without blocking other requests it can process your request.

\* synchronous → code is executed line by line in the order it appears. Each line waits for the previous one to finish before running.

Syntax:

```
fetch("url").then(function(response) {
```

return response.json();

```
}).then(function(data) {
```

... you can use data ...

```
).catch(function(error) {
```

... display exception ...

```
).finally(function() {
```

... executes always ...

```
)
```

"fetch()" promise.

"async" methods.

Requests it can process

each line waits for the

previous one to finish before running.

fetch() is a built-in Javascript function used to :

\* Request data from APIs/Server

\* Sends data to servers (PUT, POST, DELETE)

\* Handle responses (JSON, TEXT, FILE)

\* It returns a promise, so it works with .then() or async/await.

## JSON :

~ ~

JSON stands for Javascript Object Notation.

Ex:

{

  "name": "Kiran",

  "age": 20,

  "gender": "male",

  "skills": ["java", "html"]

}

Syntax:

  "key": "value"

\* JSON is a lightweight data format used for storing and exchanging data, especially between a client & server.

\* Keys are always strings (in double quotes)

\* used for sending the data b/w Frontend and Backend

\* Easy to write and read

\* Lighter and faster than XML and Language independent (used in Java, Python...).

`JSON.stringify()` : To convert Object → JSON string

`JSON.parse()` : To convert JSON string → Object

Object methods :

1) `Object.keys()` : Return array of all keys in the object.

Syntax: `Object.keys(Object)`

2) `Object.values()` : Return array of values.

Syntax:

`Object.values(Object)`

3) `Object.entries()` : Returns key-value pairs in array format.

Syntax:

`Object.entries(Object)`

4) `Object.hasOwnProperty()`

checks if a key exist in an object

Syntax:

`Object.hasOwnProperty(key)`

Ex: `var person = {`

`name: "thiru",`

`age: 21`

`}`

The members of object are accessible within the object using "this" keyword.

`person.name`

The members of object are accessible outside the object by using "person.age".

FAQ)

i) How to Read an keys from object?

By Using `Object.keys(ObjetName)` (or) `for in loop`

ii) How do we know the data type of value in key?

By using "typeof" operator.

\* A Map Object cannot contain the duplicate keys and can contain duplicate values.

### Map Methods:

1) set(): It adds or updates the key-value pair to map object

Syntax:

`mapObj.set(key, value);` Ex: `Map.set("name", "Thru");`

2) get(): It returns the value of specified key.

(or)  
get(key): Returns the value for given key.

Syntax:

~~mapObj.get(key);~~ `mapObj.get(key);` Ex: `m.get("name");` // thru

3) delete(): Removes the key-value pair

Syntax:

`mapObj.delete(key);`

4) clear(): Removes all items from the map

Syntax:

`mapObj.clear();`

5) keys(): Returns all keys

`mapObj.keys();`

6) values(): Returns all values

`mapObj.values();`

7) entries(): It Returns the Key-Value pairs

Syntax:

`mapObj.entries();`

8) size(): It Returns the Count of keys

Syntax:

`mapObj.size();`

9) has(): Check if key exists.

Syntax:

`mapObj.has();`

## Date and Time

---

- \* In Javascript, Date is the built-in object used to work with dates and time.
- \* It is a non-primitive (object) datatype.
- \* We can create date using new Date() constructor.

Ex: `let now = new Date();`

### Creating Date Objects

#### 1) New Date()

Creates a date object with current time & date.

#### Syntax:

```
let now = new Date();
```

#### 2) new Date(year, month, day, hours, min, sec, ms)

Month starts from 0 to 11  
 January                      December

#### Syntax:

```
let d = new Date(2025, 0, 26, 10, 30, 0);
```

#### 3) new Date(dateString)

Creating a date from a string format

#### Syntax:

```
let d = new Date("2025-11-26");
```

#### 1) toLocaleDateString()

\* Return only the date

\* Formatted based on Country / Locale like India, US, UK etc

#### Syntax

```
date.toLocaleDateString(locale, options);
```

Ex: `now.toLocaleDateString("en-IN") ; // 26/11/25`

### Important Date methods (Get methods)

1) getFullYear() → Returns year  
`dateObj.getFullYear();`

2) getMonth() → Returns integer value between 0 and 11  
`dateObj.getMonth();`

3) getDate() → Returns <sup>integer value b/w 1 to 31</sup> of month (0-31)

4) getDay() → Returns integer value between 0 and 6

5) getHours() → Returns integer value b/w 0 and 23

6) getMinutes() → Returns integer value b/w 0 and 59

7) getSeconds() → Returns integer value b/w 0 and 60

8) getMilliseconds() → Returns integer value b/w 0 and 999

### 2) toLocaleTimeString()

- \* Returns only the time.
- \* with hours, minutes, seconds in Local time format.

Syntax:

```
date.toLocaleTimeString(locale, options);
```

Ex:- now.toLocaleTimeString("en-IN"); // 10:25:30 PM.

### 3) toLocaleString()

- \* Returns both date and time together.

- \* Combination of date + time.

Syntax:

```
date.toLocaleString('Locale', options);
```

Ex:- now.toLocaleString("en-IN"); // 26/11/2025, 10:25:30 PM.

Note:- set is used to update/modify date and time values in a Date object.

#### 1) setHours()

sets the hour (0-23)

Syntax:

```
date.setHours(hours);
```

#### 2) setMinutes()

sets the minutes (0-59)

```
date.setMinutes(minutes);
```

#### 3) setSeconds()

sets the seconds (0-59).

#### 4) setMilliseconds()

sets the milliseconds (0-999)

Ex:-

a.setHours(15); // sets 3 pm

a.setMinutes(45); // sets 45 minutes

a.setSeconds(30); // sets 30 seconds

a.setMilliseconds(500); // 0.5 seconds

1 sec = 1000 ms.

a.setDate(15); // sets 15<sup>th</sup> day of month

a.setMonth(0); // January

#### 5) setDate()

sets the Date (0-31)

#### 6) setMonth()

sets Month (0-11)

Jan Dec

\* The above functions are used to change Date and Time.

## JavaScript Timer Events

1) setTimeout():  
 \* setTimeout() is used to execute a function only once after a specified time

Syntax:

```
setTimeout(function() {}, interval);
```

Ex: setTimeout(function() {

console.log("Hello after 3 seconds");

}, 3000); // Executes only one time after 3 seconds

2) setInterval():

\* setInterval() executes a function repeatedly at regular time intervals.

Syntax:

```
setInterval(function() {}, Interval);
```

Ex: setInterval(function() {

console.log("Runs every 2 seconds");

}, 2000);

3) clearTimeout(): It is used to stop/cancel a scheduled setTimeout()

before it runs.

Syntax:

```
clearTimeout(timeoutReference);
```

4) clearInterval(): used to stop a running setInterval()

Syntax:

```
clearInterval(intervalReference);
```

## JavaScript Math Functions

javascript provides a built-in Math Methods that contain many mathematical methods.

- 1) **Math.round()**: It Returns closest integer value of given number

Syntax:

```
Math.round(number);
```

Ex: `Math.round(4.6); // 5`

- 2) **Math.floor()**: It Returns the down to the nearest integer

Ex: `Math.floor(4.9); // 4`

- 3) **Math.trunc()**: Removes the decimal part and Returns integer only.

Ex: `Math.trunc(7.89); // 7`

- 4) **Math.max()**: It Returns maximum value of the given number

Ex: `Math.max(n1, n2, n3, ...);`

Ex: `Math.max(10, 20, 30); // 30`

- 5) **Math.min()**: It Returns minimum values of the given number

Syntax:

```
Math.min(n1, n2, n3, ...);
```

Ex: `Math.min(10, 20, 30); // 10`

- 6) **Math.pow()**: It Returns value of base to the power of exponent

Syntax: `Math.pow(base, exponent)`

Ex:- `Math.pow(2, 4); => 24 = 16`

- 7) **Math.sqrt()**: It Returns the square root of the given number

Syntax:

```
Math.sqrt(number);
```

Ex:- `Math.sqrt(49); // 7`

- 8) **Math.abs()**: It Returns the absolute value (positive);

Syntax:

```
Math.abs(number); Ex: Math.abs(-50); // 50
```

- 9) **Math.random()**: It Returns Random number between 0 and 1

Syntax: `Math.random(); Ex: Math.random(); // 0.12345`

`Math.sin(), Math.cos(), Math.tan();` It Returns the values of trigonometric values.

### JavaScript Comments

# Lines of text that are ignored by the Javascript engine during code execution are called comments

#### Syntax:

1) single-line comment

// Comment text

#### 2) Multi-line comment

```
/*
comment line-1
comment line-2
*/
```

### JavaScript Operators

#### 1) Arithmetic Operators:

+ (Addition) :  $10 + 5 = 15$

- (Minus) :  $10 - 5 = 5$

\* (Multiplication) :  $10 \times 5 = 50$

/ (Division) :  $10 / 5 = 2$

% (Modulus/ Remainder) :  $10 \% 3 = 1$

\*\* (Exponent/power) :  $2 ** 3 = 8$

++ (Increment) :  $x++$

-- (Decrement) :  $x--$

#### → pre-increment

$++x$

value increased first, then use,

$a = 5$

$b = ++a;$

`console.log(a); // 6`

`console.log(b); // 6`

#### Post-increment

$x++$

value used first, then increased

$a = 5$

$b = a++;$

`console.log(a); // 5`

`console.log(b); // 6`

#### 2) Comparison Operators:

Comparison Operators are used to compare two values and return a Boolean (true or false).

`== (Equal to)` :  $2 == 2$

`!= (Not Equal to)` :  $2 != 3$

`> (Greater than)` :  $2 > 3$

`>= (Greater than or equal to)` :  $2 \geq 3$

`< (Less than)` :  $3 < 2$

`<= (Less than or equal to)` :  $3 \leq 2$

`== (strictly equal to)` :  $2 === 2$

FAQ) What is difference between  $\text{!=}$  and  $\text{==}$  ?

$\text{!=}$  : can compare values of various types

$\text{==}$  : can compare only identical types of same type

Ex: `Var x = 10;`

`Var y = "10";`

$x == y$ ; // true

$x != y$  // false

### 3) Assignment Operators:

Assignment operators are used to assign the value to the variable.

$=$  (Assignment) :  $a = b + c$

$+=$  (Assignment Add) :  $a = a + b$  ||  $a += b$

:  $a -= b$  ||  $a = a - b$

$*$  =

:  $a *= b$  ||  $a = a * b$

$/=$  :  $a /= b$  ||  $a = a / b$

$\% =$  :  $a \% = b$  ||  $a = a \% b$

### 4) Logical Operators:

$\&\&$  (Logical AND) : Returns true only if both conditions are true.

Syntax:

Condition1  $\&\&$  Condition2;

|| (Logical OR) : Returns true if at least one condition is true

$\|$  (Logical OR) : Returns true if at least one condition is true

Syntax:

Condition1  $\|$  Condition2;

! (Logical NOT) : It is used to Reverse of boolean value.

Syntax: `!condition`

true  $\rightarrow$  false  
false  $\rightarrow$  true

## 5) Conditional (Ternary) operator:

shortcut for if-else

Syntax:

Condition ? valueTrue : valueFalse;

Ex:

let age = 20;  
age >= 20 ? "Adult": "Minor";

## 6) Special Operators

a) typeof Operator: It is used to find the datatype of variable.

Syntax:

typeof Variable

b) instanceof operator: Checks whether an object belongs to particular class

Syntax:

Object instanceof Constructor

Ex: let d = new Date();  
console.log(d instanceof Date); // true  
console.log(d instanceof Array); // false

c) new operator: Used to create an instance of object

Syntax

new ConstructorName();

Ex: let ob = new Object();

let arr = new Array();

d) in Operator: Checks whether a property exists in an object or array.

Syntax:

property in object

Ex: let person = { name: "John", age: 21 };  
console.log("name" in person); // true  
" " ("Salary" in person); // false

e) delete Operator: Removes a property from an object.

Syntax:

`delete Object.property;` Ex: `delete student.age;`

f) Spread Operator: Expands elements of arrays, objects, strings

Syntax:

`let arrayName = [.... value1, value2.] or [.... ExistingArray, value1, value2];`

Ex: `let arr = [1, 2, 3]` or `let arr = [.... 10];`

`let newArr = [...arr, 4, 5]`

g) Rest parameter: Used to collect (packs) multiple arguments into a single array

Syntax:

`-function functionName(... parameterName) {  
 // parameterName becomes an Array.  
}`

Ex: `function sum(...num) {  
 return num.reduce((a, b) => a + b);  
}  
console.log(sum(10, 20, 30)); // 60`

h) void : evaluates an expression and always returns undefined, no matter what expression produce

Syntax:

`void expression`

Ex: `console.log(void 0); // undefined`

`console.log(void (10 + 12)); // undefined`

## Javascript Statements

1) Selection statements are used to make decisions in a programme.

- a) if
- b) if else
- c) switch

### a) if statement

Executes a block of code only when a condition is true.

Syntax:

```
if (Condition) {
    // Code to run if condition is true
}
```

Ex: let age = 20

```
if (age >= 18) {
    console.log("you are the Major");
}
```

### b) if...else Statement

When you want to execute one block if true and another block if false

Syntax:

```
if (Condition) {
    // Code if condition is true
}
else {
    // Code if condition is false
}
```

Ex: let marks = 45;

```
if (marks >= 35) {
    "Pass";
}
else {
    "Fail";
}
```

### c) if...else...else statement

Used when you need to check multiple conditions.

**Syntax:**

```

if (Condition-1) {
    // Code block to be executed if Condition-1 is true
}

else if (Condition-2) {
    // Code block to be executed if Condition-1 is false and Condition-2 is true
}

else {
    // Code block to be executed if all pending conditions are false
}

```

**d) Switch Statement**

- \* Switch statement is used when you want to compare one variable with many possible values.
- \* When one value needs to be checked against multiple conditions.
- \* It is easier and cleaner than writing multiple if...else-if statements.

**Syntax:**

```
switch (expression) {
```

**Case value1:**

```
    // Code to execute if expression == value1
    break;
```

**case value2:**

```
    // Code executes if expression == value2
    break;
```

....

**default:**

```
    // Code if no case matches
}
```

Ex: let day = 3;

```
switch (day) {
```

case 1 : sunday

```
    console.log('Sunday');
```

break;

case 2 :

Monday;

break;

case 3 :

Tuesday;

break;

case 4 :

Wednesday;

```

break;           // Tuesday
default:
    console.log("Invalid Day");
}

```

### a) Looping statements

- \* Loops are used to repeat a block of code multiple times until condition becomes false.

#### a) for loop

- \* used when the number of iterations is known

Syntax: ①      ②      ④  
 for (initialization; condition; increment/decrement)

// Code to Run  
③

}

#### b) while Loop

- \* Used when the number of iterations is unknown
- \* It is executed as long as the specific condition is true

Syntax:

```

while (condition) {
    // Code to Run
}

```

#### c) do..while loop

Executes the code at least once (even if the condition is false)

Syntax:

```

do {
    // Code to Run
} while (condition);

```

semicolons mandatory on  
white.

Ex: var i=1  
 do {  
 console.log(i);      //  
 i++;
 } while (i >= 5);

### d) for...in loop

\* Used to iterate over object properties. Var student = { name: "thiru", age: 20 };

Syntax:

```
for (var key in object)
{
    // Code
}
```

Ex: for (var key in student)
{
 console.log (student[key]);
}
 ↴ ↴
 name
 age

### e) for...of loop

\* Used to iterate over iterables (arrays, strings, sets, maps).

Syntax:

```
for (var value of iterable) {
    // Code
}
```

Ex: var num = [10, 20, 30]
for (var n of num) {
 console.log (n);
}

### f) forEach loop

\* forEach() is an array method used to execute a function once for every element in the array.

\* It is used for only arrays (not objects).

Syntax:

```
array.forEach(function(item) {
    // code
});
```

Ex: var arr = ["thiru", "shiva", "mahi"];

```
arr.forEach(function(value) {
    console.log (value);
});
```

### 3) Jumping Statements:

#### a) break

Used to stop (terminate) a loop or switch statement immediately.

Syntax:

```
break;
```

#### b) continue

Skips the current iteration and continues with the next iteration of the loop.

Syntax:

```
Continue
```

#### c) return

Used inside a function to stop function execution and optionally return a value.

Syntax:

```
return value;
```

## JavaScript Functions:

A Function in JavaScript is a block of code designed to perform a specific task.

You can define a function once and reuse it many times.

\* Refactoring Code

\* Removes Code Duplication

\* Improving Reusability

Refactoring → Refactoring is the process of extracting a set of statements and placing them inside the function so that the same code can be reused multiple times.

JavaScript allows 2 ways to create a function

i) Function Expression

A function stored inside a variable

Syntax:

```
Var f = function (parameter) {
    // Code }
```

## 2) Function Declaration (Named Function)

Syntax:

```
-function functionName (parameters) {
    // Code to execute
}
```

Ex:

```
function greet() {
    console.log("Hello Javascript");
}
greet(); // Calling the function
```

## Arrow Function : (ES6)

\* It is a short and technique for writing a function expression.

Syntax: → var functionName = () => { };

var functionName = (param) => { }

// Code  
};

var functionName = (param1, param2) => {  
 // Code  
};

→ ( ) → function and its params

{ } → return and definition

Ex: var addition = (a, b) => {  
 return a+b;  
};

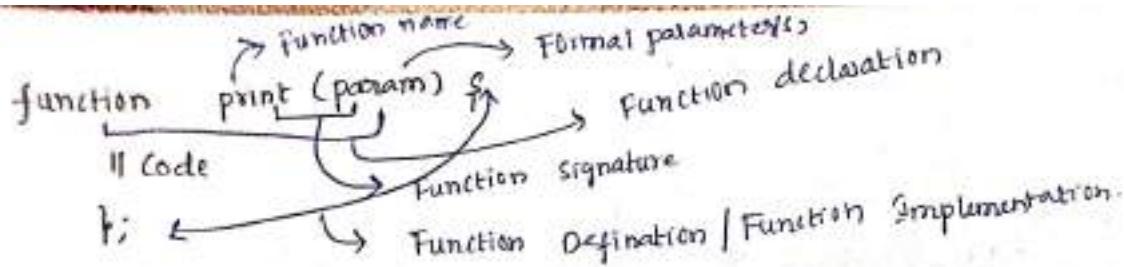
var addition = (a, b) => a+b;

## Function Recursion:

Function is a technique where a function calls itself until a specific condition is met.

Syntax: function functionName() {
 // recursive call
 functionName();
}

Ex: -function factorial (n) {
 if (n == 1) { return 1; }
 return n \* factorial (n-1);
}
console.log (factorial (5)); // 120



### Formal Function Parameters (Inside the function)

\* these are the placeholders/variables inside the function definition

Ex:

```
function greet(name)
{
    console.log("Hello " + name);
}
```

Actual Parameters (values passed) (Real values passed to the function when calling it.)

```
greet("Thiru");
```

\* A function parameter can be any type

- primitive
- non-primitive
- function

Ex: name('Thiru'), name(1), name(true), name({}), name({}), name(function{})

### Multiple Parameters

\* Multiple parameters can accept any number of parameters

\* Each parameter is comma-separated inside the parentheses

### Syntax:

```
function functionName(param1, param2, param3...)
```

```
{
    if code
}
```

\* Every parameter is required. Order dependency

Ex: `function details (name, age);`

```
{
}
details ("thisu", 20); ✓
details (20, "thisu"); X
```

\* If you want to ignore any specific parameter, then you have to handle using

Undefined type.

\* ECMAScript allows max 1024 parameters.

\* ES5+ version introduced "rest" parameters.

### Rest Parameters (...param)

\* Define using ... paramName

\* Rest Parameter is always of array type

\* Rest Parameter can have only one rest parameter.

\* Rest Parameter must be the last parameter in the function

Ex: `function sum (...num){`

`return arr;`

`}`

`sum (10, 20);`

Syntax:

`function demo (...param1);`

### Spread Parameters

\* A Spread allows an array value to expand into multiple individual arguments.

\* It can be array type.

Syntax:

`functionName (...values);`

Ex:- `demo (...values);`

Ex:- `let arr = [10, 20, 30, 40];`

`function add (a, b, c){`

`return (a + b + c);`

`}`

`add (...arr);`

Note:

(1) Rest Parameter is about formal parameters, a single formal parameter allows multiple arguments

(2) Spread Parameter is about, a single actual parameter can spread into multiple formal parameters.

## Function Promise

- \* A Promise is an Object that handles asynchronous operations.
- \* It represents the value that will be available now, later, never.
- \* Asyn = Is an unblocking technique that allows the task to perform without blocking other tasks.
- 1) pending
- 2) resolved (fulfilled) = defines then()
- 3) Rejected  $\Rightarrow$  defines catch()

Syntax:

```

var defName = new Promise(function(resolve, reject) {
    if (Condition) {
        resolve "success";
    }
    else {
        rejected "failure";
    }
});
defName.then(function(){}).catch(function(){});
```

Function Return:

- \* Sends a value back to the calling code
- \* Stop / terminate the function immediately
- \* Use function output somewhere else

Syntax:

```

function functionName() {
    return value;
}
```

What is callback?

- A callback is a function passed an argument to another function

Syntax:

```

function main(callback) {
    callback(); // calling function passed an argument
```

## Exceptional Handling

28-11-25

- \* Exceptional handling is a technique, which is to avoid abnormal termination of application.

- What is Error?
- \* A error is a problem in the code that occurs at runtime or compile time and stops the programme execution
  - \* Errors are thrown by Javascript Engine

Ex:

- SyntaxError → Wrong syntax;
- ReferenceError → Using undefined variables
- TypeError → Invalid operations on a type
- RangeError → Out of Range

Ex:-

`let x = y + 10;`

`y is not defined → ReferenceError`

What is Exception?

- \* An exception is a runtime event that disturbs the normal flow of execution but can be handled using exception-handling mechanisms.

### 1) Try

- \* Try is a keyword is used to defines block of code to test exceptions.
- \* Try is a keyword not crash if an error occurs.
- \* So, the programme not crash if an error occurs.
- \* Try defines a monitoring block.

Syntax:

```
try {
    // doubtfull code
}
```

## 2) catch

\* catch is keyword is used to handle exception that occur in the try block.

Syntax:

```
try {
    // Code that might throw an error
}
catch (error) {
    // Code to handle error
}
```

## 3) throw

\* throw is a keyword, which is used to manually generate (throw) an exception.

Syntax:

```
throw expression;
```

## 4) finally

\* finally is a keyword, which will executes always, whether an exception occurred or not in the try block.

\* It is used for to perform cleanup actions.

Syntax:

```
try {
    // Code that may throw an error
}
catch (error) {
    // Code to handle the error
}
finally {
    // Code always executes (cleanup)
}
```

## Javascript OOPs

\* OOPs = Object Oriented Programming System

\* Features (Advantages)

- Code Reusability
- Code Separation
- Code Extensibility
- Code Security

\* Drawbacks

- No direct low-level memory access
- Cannot access internet with hardware directly
- Takes more memory
- Complex configurations
- Tedious for small tasks

\* Evolution of OOP

1960 - Alan Kay introduced the idea of 'objects' in programming  
 1967 - John Olay & Kristian Nygaard created SIMULA 67, first true OOP lang.  
 1970 - Trygve Reenskaug introduced MVC (Model-View-Controller) using Smalltalk  
 1975 - C++  
 1990 - Java

### Javascript Module

A module is a set of variables, functions or classes that can be reused anywhere

\* Benefits:

- Code Reusability
- Extensibility
- Ability to create Javascript Libraries

\* ES Modules (ESM) → default in browsers.

\* Official Javascript Standard.

Creating a Module:

- 1) Creating folders: src/library/modules
- 2) Add file: home.module.js
- 3) Add variables, functions, classes
- 4) Use to export to make members available to other files

## export Keyword:

- \* export is a keyword, to share variables, functions, classes from one javascript file, so that other files can use them

### Syntax:

```
export const name = "John";
export function greet() {
    return "Hello";
}
export class Student {
```

\* planned export

- \* Every module can have one "default" export

## import Keyword:

- \* import is a keyword, to access variables, functions, classes from another file to current file.

### Syntax:

```
import defaultMember, {other} from "moduleName";
import {greet} from ".../name.module.js";
```

\* Name exports use {}  
\* Default exports ~~use~~ does not use {},

- Ex: import welcome, {greet} from ".../name.module.js";
- \* A class is a template provides sample data and logic which you can customize and implement according to requirements  
(or)
- \* A class is a template or Blueprint used to Create Objects.
- \* Class Contains properties(data), methods (functions).

### i) Class Declaration

```
class className {
    // properties
    // Methods
}
```

### ii) Class Expression

```
const className = class {
    // properties
    // constructor
    // methods
}
```

## Creating an object (Instance)

let ObjName = new Classname();

new is a keyword, to allocate the memory

## Class Members

- a) Property
- b) Accessor
- c) Constructor
- d) Method

### FAT

1) Can we define variable or function as class members?

No

2) Can we define a variable or function in class?

Yes

3) Why function and variables are not allowed as class members?

Because, All class members variable and function are immutable.

A class member are always must be mutable.

hence, Variable and function are not class members.

### a) property

\* property is a key-value pair attached to an object

\* properties are variables (data / values) that belongs to object or class instance

\* properties are immutable which we can change the behaviour dynamically.

\* property is a mutable which we can change the behaviour dynamically.

Ex: let product = {

```
  name: "Rahul",
  age: 25,
  city: "Kushal"
};
```

} properties.

### b) Accessors

\* Accessors are to used to fine grained control over the property

\* Accessors = Getters + Setters

#### i) Getter

To Read property

```
get propertyName() {
```

returns value

}  
 >> Setter  
 to write/update property  
 set propertyName() {  
 || validation  
 }

### c) Method

\* A method is a function that is a property of an object or class

\* Methods defines Behaviour for Objects

\* Methods are Mutable

\* They can access objects properties using "this". and are called with parenthesis.

#### Syntax:

```
class Method:  
class className {  
    methodName(parameters)  
    {  
    }  
}
```

#### Object Method

```
let ObjectName = {  
    property1: value1,  
    property2: value2,  
    methodName()  
    {  
        // code here  
    }  
};
```

### d) Constructor

\* A Constructor is a special Method in a class that is automatically called when an new object is created

\* Constructor is defined using the keyword Constructor inside the class

#### Syntax:

```
class className {  
    constructor(parameters)  
    {  
        this.parameter1 = parameter1;  
        this.parameter2 = parameter2;  
    }  
}
```

```
let obj = new className(param1, param2, ...); // Object creation
```

\* Only one constructor method is allowed in a class.

\*\*

Code Reusability and Extensibility :

We can handle the code Reusability and Reusable Extensibility by using 2 technique.

a) Inheritance

\* Inheritance is the process of accessing members of one class in another class by Configuring Relation between classes.

(Or)

+ Inheritance is the process of one class (<sup>↳ derived class</sup> / child / subclass) inherits properties and methods from another class (<sup>↳ parent / superclass / Base class</sup>)

\* Represents "Is-A" Relationship

\* Code Reusability

Syntax:

```
class className {  
    // properties & methods  
}  
  
class subclass extends superclassName {  
    // properties & methods  
}
```

\*\* extend keyword

\* extend is a keyword, used for a class to inherit properties and methods from another class.

\*\* Super keyword

\* Super keyword is used in a child class to refer the parent class.

\* call parent constructor

\* Call parent methods

### b) Aggregation

- It is the process of accessing members of one class in another class without configure any Relation between classes
- \* 'Has-A' Relationship
- \* It is also known as 'Object to Object' Communication.

### c) polymorphism:

- poly = many } Many forms
- morph = forms }
- \* polymorphism is the ability of class or object can use the memory of multiple derived classes.
- EX:- A person can speak telugu, hindi, English etc.

Ex: class Animal {

```
    speak() {
        console.log("Animal makes sound");
    }
}
```

class Dog extends Animal {

```
    speak() {
        console.log("Dog makes sound");
    }
}
```

class Cat extends Animal {

```
    speak() {
        console.log("Cat makes sound");
    }
}
```

```
const animal = [new Animal(), new Dog(), new Cat()];
```

```
animal.speak();
```

here some speak method name behaves differently depending on the object type  
that is the polymorphism

## Event Handling in Javascript

What is Event ?

An event is a message / notification sent by a sender (HTML element) to a subscriber (Javascript function) when something happens on the web page.

Ex:- click, keypress, mouseover etc.

\* Javascript event handling follows a software design pattern called

Observer Pattern.

• sender (subject): Elements trigger events

• subscriber (observer): Function that listens for the event.

Communication flow :-

- 1) sender triggers an event (eg: click)
- 2) subscriber receives notification
- 3) subscriber executes actions.

Syntax:

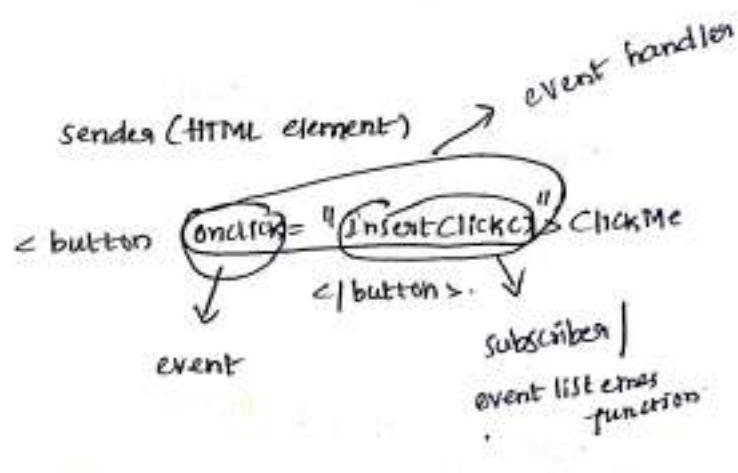
subscriber (Javascript function)

-function Insertclick()

{

// actions to perform

}



\* Javascript event handlers receive 2 default arguments.

a) this

b) event (or e)

## 1) Mouse Events

a) onclick : Triggered when the user single-clicks on an element

syntax :

onclick = "functionName();"

```
Ex: <button onclick="showMsg();">
    click me </button>
    <script>
        function showMsg() {
            alert('button clicked');
        } </script>
```

b) ondblclick : Triggered when the user double-clicks an element

syntax :

ondblclick = "functionName();"

c) onmouseover : Triggered when the mouse pointer moves over an element

syntax :

onmouseover = "functionName();"

d) onmouseout : Triggered when the mouse pointer moves out-of an element

syntax :

onmouseout = "functionName();"

e) onmousemove : Triggered continuously when the mouse moves inside the element.

syntax :

onmousemove = "functionName();".

f) onmousedown : Triggered when the mouse button is pressed down (left/right button)

g) onmouseup : Triggered when the mouse button is released after pressing

syntax :

onmouseup = "functionName();"

10) oncontextmenu : Fired when the right-mouse button clicked on an element

h) Keyboard Events : When user can interacts with keyboard.

a) onkeydown : Triggered when the user presses a key down (fires immediately)

syntax :

onkeydown = "functionName(event);"

b) Onkeypress : Triggers when the key pressed AND character is typed

Syntax:

Onkeypress = "functionName (event)"

c) Onkeyup: Trigger when the user releases the key.

Syntax:

Onkeyup = "functionName ()"

### 3) Element State Events:

a) Onchange : Fired, when the value of an element changes and the element loses focus

Syntax:

<element onchange = "functionName (this, event)"></element>

b) Onblur: Fired, when an element loses focus.

Syntax:

Onblur = "functionName (this, event)"

c) Onfocus: Fired, when an element gets focus.

\* used to highlight active input

Syntax:

Onfocus = "functionName (this, event)"

### 4) Clipboard Events:

a) Oncopy : Triggered when the user copies text ( Ctrl + C or right click → copy )

Syntax:

Oncopy = "functionName ()"

b) Oncut : Triggered when the user cuts ( Ctrl + X or right click → cut )

Syntax:

Oncut = "functionName ()"

c) ~~oncopy~~ onpaste : triggers when the pasted content (ctrl+v or right-click → paste)

Syntax:

onpaste = "functionName()"

### 5) Form Events

a) onsubmit : Triggered when a form is submitted. Used to validate data or prevent default submission.

Syntax:

onsubmit = "functionName()"

b) onreset : Triggered when a form is reset using a reset button

Syntax:

onreset = "functionName()"

### c) Timer Events

a) setTimeout() : To execute a function only once after a specified time

Syntax:

setTimeout (function() {}, interval);

b) clearTimeout() : to stop/cancel a scheduled setTimeout() before it begins to run

Syntax:

clearTimeout (TimeOut Reference);

c) setInterval() : executes a functionality at regular time intervals.

Syntax:

setInterval (function() {}, Interval);

d) clearInterval() : to stop a running setInterval()

Syntax:

clearInterval (Interval Reference);

e) scroll events → When a page or an element is moved up, down, left, right

onscroll : Fired when page or element is scrolled

Syntax:

onscroll = "functionName()"

## 6) Browser Events

a) Onload : Fires the When the entire page (HTML, CSS, Javascript.) is fully loaded

Syntax:

Onload = "functionName()"

b) Unload : Fires when the user leaves the page or close the browser tab.

Syntax:

unload = "functionName()"

c) onbeforeunload : Fired just before

the page unloads - can show a warning

if the user tries to leave

Syntax:

onbeforeunload = "functionName()"

## \*\* addEventListener()

\* addEventListener() is a method in Javascript used to attach an event handler to an element, document, window etc.

\* It allows dynamically to configure events

\* It makes clean separation of Code and UI.

Syntax:

element.addEventListener( event, function() {} );

\* Removing Event Listener

Syntax

Element.removeEventListener  
(eventname, functionName);

Note:

event write in addEventListener  
like below:  
"onClick" X  
"click" ✓

Ex:-

```
document.getElementById('btn').addEventListener("click", function() {
    alert('Button clicked');
});
```

FAQ) how to change events for elements dynamically ?

By adding event listener.

## State Management in Web Applications

- \* Every application uses Http orHttps as a protocol
- \* Http is a stateless
  - Stateless → Cannot Remember information pages or Requests
- \* Http - Hyper Text Transfer Protocol
  - ↳ used for transferring data b/w client (browser) and server
  - default port: 80
- \*Https → Hyper Text Transfer Protocol secure
  - ↳ Https is Http with encryption
  - ↳ provides authentication
  - default port: 443
- \* Web applications implement state management techniques to store and access across pages or Requests.
- + State Management is classified into 2 types:
  - 1) Client-side State Management
  - 2) Server-side State Management
- 1) Client-side State Management
  - + stores data on the Client device (browser)
  - techniques:
    - a) Query String
      - \* Data is appended in the URL with a key-value format.
      - ? key = value
    - \* Access: location + search

### b) Local storage

- \* Local storage is permanent storage.
- \* Local storage is available even after your device shutdown.
- \* Local storage is accessible all tabs in the same browser.
- \* You have to delete manually from browser memory.
- \* We can store 10mb of data in LocalStorage.

Syntax:

```
// stored data
localStorage.setItem("username", "thiru");

// Retrieve data
var value = localStorage.getItem("username");

// Remove data
localStorage.removeItem("username");

// clear all Local storage
localStorage.clear();
```

### c) Session Storage

- \* Session storage is temporary storage.
- \* Data is removed when the tab or browser is closed.
- \* It is not accessible to other tabs.
- \* We can store 10mb of data in Session storage.

Syntax:

```
// Store data
sessionStorage.setItem("username", "thiru");

// Retrieve data
sessionStorage.getItem("username");

// Remove data
sessionStorage.removeItem("username");

// clear all session storage
sessionStorage.clear();
```

## d) Cookies

- \* Small text files stored in the browser.
- \* Can store client-related information.
- \* Can be temporary (session cookies) or permanent or persistent (with expiry).
- \* Browser must have Cookies enabled.

Syntax:  
`navigator.cookieEnabled;`      || true if cookies are enabled

Syntax:  
 || set a cookie  
`document.cookie = "username = thinu"; expires = Fri, 31 Dec 2025 23:59:59;"`

|| Read all cookies  
`var Cookies = document.cookie;`

|| Delete a cookie (set expiry to past date)  
`document.cookie = "username = ; expires = Thu, 01 Jan 2027 23:40:40;"`

Attributes  
`expires:` specify the exact date/time when the cookie should expire

`max-age:` specify the lifetime of the cookie in seconds

`path:` defines the URL path where the cookie is accessible

### Note:

- \* LocalStorage and sessionStorage and Cookies are stored data in the Key-Value Pairs.

Ex: `name : "Pavan";` ✓

Set

- \* Set is the one of datatype in javascript
- \* A set is off a collection of unique values.
- \* It cannot have duplicate values.
- \* It can store any type: numbers, strings, objects etc

## Syntax:

`var s = new Set();`      Ex:- `var s = new Set([1,2,3,4]);`

## Methods:

- 1) `add()` → Add values to the set
- 2) `delete(value)` → Removes a value from the set
- 3) `has(value)` → Check if value exists
- 4) `clear()` → Remove all values
- 5) `size()` → Returns the no. of elements
- 6) `forEach()` → Iterates through the set

Note:

Convert set → array.

1) `let a = new Set([1,2,3,4,4,5]);` (or)  
`let b = Array.from(a);`

|| Using spread operator  
`let arr = [...a];`  
`console.log(arr);`

## DOM (DOCUMENT OBJECT MODEL)

- DOM (document Object Model) is a programming interface for HTML and XML documents.

It Represents a web page a tree structure where each html element, attribute, text is a node.

By using dom, we can manipulate web pages dynamically

- a) Create a new Elements
- b) Add Elements
- c) Removing Elements
- d) Access page Elements (Id, class)
- e) Change content and styles dynamically
- f) Handles events
- g) Change Attributes
- i) Selecting Elements
  - a) getElementById → selects a single element with unique Id.
  - b) getElementByClassName → selects all elements with specific class
  - c) getElementByTagName → selects all elements with specific HTML tag
  - d) getElementsByTagName
  - e) Query Selector → selects the first element that matches a CSS selector
  - f) Query SelectorAll → selects all the elements that matches a CSS selector

Other useful Methods:

document.forms → Access all form on a page

document.images → Access all images

document.links → Access all links

## 2) properties to change Content

- a) innerHTML (include HTML tags)
- b) innerText
- c) textContent
- d) value

## 3) properties to change styles

- a) style

Change inline CSS styles directly

Syntax:

`element.style.property = "value";      Ex:    box.style.color = "Red";`

## 4) Creating Elements

1) `document.createElement();`

↳ Create a new HTML element but does not add it to the page yet.

Ex:-

`var el = document.createElement('div');`

## 5) Adding the Elements

a) `appendChild()`

↳ Adds the new Element at the last child of a parent Element

Syntax:

`parent.appendChild(element);`

Ex: `document.body.appendChild(div);`

b) `insertBefore()`

Add the new Element before a Reference Element

Syntax

`parent.insertBefore(newElement, referenceElement);`

### 6) Removing Elements

a) `remove()` → Removes the element directly from the DOM.

Syntax:

`element.remove();`

b) `removeChild()` → Removes the child element from the parent.

Syntax:

`Parent.removeChild(Child);`

### 7) Change Attributes

a) `setAttribute()`: Add a new attribute, (or) change the value of existing attribute

Syntax:

`element.setAttribute(attributeName, value);`

b) `getAttribute()`: Get the value of specific attribute.

Syntax:

`element.getAttribute(attributeName);`

c) `removeAttribute()`: Remove an attribute from element

Syntax:

`element.removeAttribute(attributeName);`

d) `hasAttribute()`: check if an element has an specific attribute.

Syntax:

`element.hasAttribute(attributeName);`

### 8) Handle Events

a) `addEventListener()`

`addEventListener()` allows dynamically to configure events

Syntax:

`element.addEventListener(event, function() {});`

b) `removeEventListener()`

\* Removing event listener dynamically

Syntax:

`element.removeEventListener(eventName, functionName);`

## Asynchronous Javascript

### \* Synchronous

- \* A synchronous operation is one that executes sequentially.
- \* Each statement wait for previous statements to complete before executing.
- \* In synchronous code, the programme blocks until the task is finished.
- \* Single threaded execution, no task runs in parallel

### \* Asynchronous

- \* Asynchronous operation is one that executes independently of programme flow.
- \* The programme can start a task and continue executing other code without waiting.
- \* Non-blocking execution, multiple tasks allows to run concurrently

### Promises

- \* A promise is an object representing the future result of an asynchronous operation, which can either success (Resolve) or fail (reject)
- (OR)
- \* A Promise is an object representing the eventual completion (success) or failure (reject) of an asynchronous operation.
- \* A promises will help to handle the asynchronous code.
- \* A promise can be 3 states
  - a) Pending : initial state, task not completed yet
  - b) Fulfilled (Resolve) : task completed successfully.
  - c) Rejected : task failed

Ex:

```

    // Create a promise and store in it a variable
    let mypromise = new Promise ((resolve, reject) => {
        // resolve, reject are functions provided by JS.

        let success = true;

        if (success) {
            resolve ("task completed successfully");
        }
        else {
            reject ("task failed");
        }
    });

```

- **then():**  
 \* then() is used to handle the Resolved (fulfilled) value of a promise  
 \* It runs only if the promise is successful (ie; resolve() called).  
 \* Handles the success.
- **catch():**  
 \* catch() is used to handle the errors when promise is Rejected.  
 \* It runs only if the promise is failure (ie; reject() called).  
 \* Handles the failures/errors

Ex:

```

    mypromise
        . then(result => console.log(result));           → Returns success data
        . catch(error => console.log(error));           → Returns errors
    . finally()

```

\* .finally() is method with promise the runs a piece of code Regardless

of whether the promise was fulfilled (resolved) or rejected.

→ It's used for clean up tasks.

Syntax:

myPromise

- then(result) =>

```

    {
      // handle success
    }
  
```

- catch(error) =>

```

    {
      // handle failure / error
    }
  
```

- finally(c) =>

```

    {
      // Runs in both cases (success or failure).
    }
  
```

});

async/await

1) async

async is a keyword is used for to declare a function as asynchronous.  
which means, the function will always return a promise

2) await

await is a keyword can be used inside an async function. It pauses  
the execution of the async function until the promise is Resolved or Rejected.

Ex:

```

async function getUserData() {
    try {
        let response = await fetch('https://jsonplaceholder.typicode.com/users/1');

        let data = await response.json();
        console.log('UserData:', data);

    } catch (error) {
        console.log('Error fetching Data:', error);
    }

    finally {
        console.log('Fetch operation completed');
    }
}

getUserData();

```

FAQ)  
How many ways we can handle the asynchronous code?

By using

- 1) callbacks
- 2) promises
- 3) Async/Await (modern way (Recommended)) ✓

### Set

- \* Set is one of datatype in JavaScript
- \* A set is ~~of~~ a collection of unique values.
- \* It cannot <sup>have</sup> duplicate values.
- \* It can store any type: numbers, strings, objects etc

### Syntax:

```
var s = new Set();
```

Ex:- var s = new Set([1, 2, 3, 4]);

### Methods:

- 1) add() → Add values to the set
- 2) delete(value) → Removes a value from the set
- 3) has(value) → Check if value exists
- 4) clear() → Remove all values
- 5) size() → Returns the no. of elements
- 6) forEach() → Iterates through the set

### Note:

Convert set → array.

- 1) Let a = new Set([1, 2, 3, 4, 4, 5]); (07)
- 2) let b = Array.from(a);

// Using spread operator  
Let arr = [...~~a~~]  
console.log(arr);

## REACT (Javascript + JSX)

FAQ) What is React?

React is a Javascript library used to build User Interface (UI).

React is a Open-source

React was developed by Facebook (Meta).

React latest version: 19.2.3

\* React extension is ".js" or ".jsx" (both are same)  
React uses React, Instagram, WhatsApp, Facebook, Spotify, Netflix, Pinterest etc.

for creating Single-page Application (SPA).

\* React and React.js are both same.

What is SPA?

A Single Page Application (SPA) is a web application that loads only one HTML page and updates content dynamically without reloading the whole page.

React features:

1) Component-Based Architecture

2) JSX (Javascript XML)

• write HTML code inside the Javascript

3) Virtual DOM

4) One-Way Data Binding

5) High Performance

6) React is Modular

\* React is Modular because, it allows developers to use only required libraries.

making applications lightweight, and use less memory.

\* React is not legacy.

7) React uses built-in Async methods [Implicitly synchronous]

8) Reusability

What is Reconciliation? → Reconciliation is the process by which

React updates the Real DOM by comparing old and new Virtual DOM using diff algorithm.

Challenges / Issues / problem in React:

- 1) Frequent updates
  - 2) Complex tooling
    - \* Setup can feel complex
  - 3) Not a complete framework
  - 4) Performance Issues
- FAQs
- 1) What is Virtual DOM? → It is a Duplicate copy of actual dom
  - 2) Virtual DOM is a lightweight copy of the real DOM stored in memory.
  - \* React uses it to update only the changed parts of the UI instead of Reloading the entire page
  - \* In React, the Virtual DOM updates first and after Composting changes, only required updates are applied to the Real DOM.
  - \* Virtual DOM is created automatically by React when JSX or React
  - \* Faster Performance, Better UI consistency
- 2) What is Shadow DOM?
- Shadow DOM is a browser feature that allows you to create a separate DOM and CSS encapsulation for Components, do not affect the rest of the page.
  - (OR) shadow DOM is a hierarchy of nodes in a component
- 3) What is DOM?
- Real DOM (Document Object Model) created by the browser that directly represents the HTML elements on the web page
- 4) What is JSX? → (Javascript + HTML)
- \* JSX → Javascript XML
  - JavaScript extensible Markup Language (JSX)
- \* JSX (Javascript XML) is a syntax extension for Javascript that allows us to write HTML-like code inside the Javascript mainly used in the React to build UI Components.

- \* JSX is not HTML
- \* JSX is not mandatory, but highly recommended.
- \* Supports Javascript inside {}
- \* Browser does not understand JSX
- \* Babel converts JSX to Javascript

↑ hard to Read, too much nesting  
difficult maintaining

|                                                                                                                                                                                                                         |                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <u>Ex:</u> React with JSX (Recommended) <pre>function App() {   return (     &lt;div&gt;       &lt;h1&gt; Welcome React &lt;/h1&gt;       &lt;p&gt; Hello &lt;/p&gt;     &lt;/div&gt;   ); }  export default APP;</pre> | React without JSX (Pure React) <pre>function App() {   return React.createElement(     'div', null,     React.createElement("h1", null, "Welcome       React"),     React.createElement("p", null, "Hello")   ); }  export APP;</pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### JSX Rules :

✓ All tags must be closed

Ex: <img src = "photo.png" /> ✓      <img src = "photo.png"/> ✓

✓ JSX is case-sensitive

\* Components should be ~~be~~ uppercase

\* HTML elements in Lowercase

Ex: <MyComponent />

<div> </div>

✓ Javascript must be inside {}

Ex: function App() {
 var name = "thirumalesh";
 return ( <h1> Hello {name} </h1> );
}

✓ Use className instead of class.

Ex:

```
<div className = "box" ></div>
```

✗ Use htmlFor instead of for

Ex:

```
<label htmlFor = "email" > Email </label>
```

✓ JSX will not allow individual elements in ui.  
JSX requires a container (or one parent Element/component) or fragments to handle elements.

Ex:

```
<h1>           => invalid  
  Hello  
</h1>
```

```
<div>  
  <h1>          Hello  
  </h1>  
</div>           => valid
```

✓ JSX, Event handles use CamelCase

```
<button onClick = {handleClick} > Click </button> // valid  
                                         (onClick)
```

```
<button onclick = {handleClick} > Click </button> // invalid.
```

✓ Inline styles uses Javascript Objects

```
<div style = {{color: "red", background-color: "blue"} } > </div>
```

✗ JSX comments use /\* \*/

✗ Boolean & null values are not rendered

|           |     |            |
|-----------|-----|------------|
| { true }  | { } | // invalid |
| { false } |     |            |
| { null }  |     |            |

FAQ)

What is Babel? → converts JSX into Javascript

Babel is a Javascript Compiler that converts ~~javascript~~ or JSX

Code into plain (or) backward compatible Javascript that the browser can understand.

\* Here, Browser does not understand the JSX (or) modern Javascript (ES6+)  
→ then we will <sup>use</sup> babel compiler for to convert the JSX into plain Javascript  
for Run the code in browser.

Ex: JSX code

```
const name = "Thirumalai";
const element = <h1>Hello {name}</h1>
```

\* the above code Babel Converts JSX into Javascript that browser can understand.

```
const name = "Thirumalai";
const element = React.createElement(
  "h1", null, "Hello", name);
```

FAQ)

Q) How many we can create react application?

a) Manually setup

```
npm install react --save
```

```
npm install react-dom --save
```

```
npm install babel/standalone --save
```

react : It is the core library for React

react-dom : It is handle virtual DOM

babel : It is Javascript compiler used by React

b) Create React App (CRA)

\* It is an official tool of React which sets up a new project with all necessary configurations automatically.

\* Pre-configured Webpack & Babel

\* Supports .JSX and ES6+ automatically

How to create a React App with CRA

`npx create-react-app foldername projectname`

Ex:- `npx create-react-app myfirstproject`

Start the development server

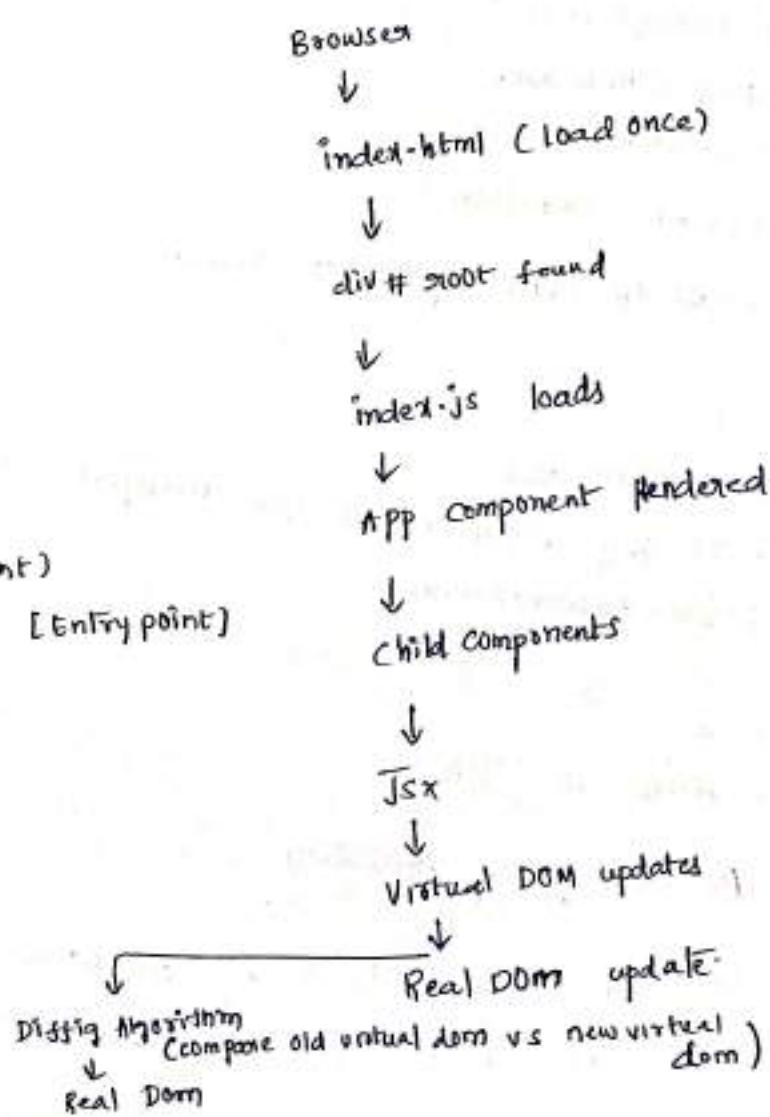
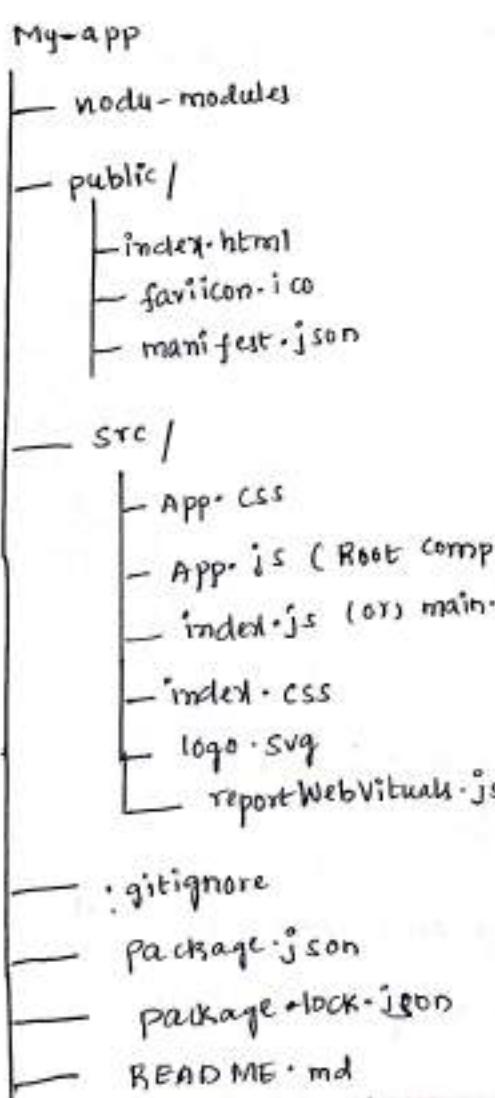
`npm start`

↳ Opens the application in browser: `http://localhost:3000`

default port in  
React.

CRA Folder Structure:

### Flow of React



## 1) node-modules :

- Contains all installed dependencies (react, react-dom, babel etc)
- Automatically created by npm

## 2) public :

Contains the static files.

Ex: html, images, documents, videos etc

## 3) src :

Contains the dynamic files (js, css, JSX etc)

- Contains all React Code here

## 4) .gitignore :

Files/Folders not to include while publishing on GIT

Ex: node-modules.

## 5) package.json :

- project meta data
- Dependencies

## 6) package-lock.json :

- Locks the exact dependencies versions.

## 7) README.md :

- It is help documentation for developers.
- project documentation

## FAQ

## 1) What is SPA ?

SPA → Single Page Application.

SPA → the browser loads only one HTML page once, and after that the content changes dynamically without reloading the whole page.

(Q8)

- index.html loads only once
- Whole page does not reload.
- Only the required React Components change (Q8) Re-Render dynamically.

\* Content changes dynamically

Components in React

\* Components must be extension with .js or .jsx (filename)

\* Components are the building blocks of the UI.

\* Component is a reusable piece of UI.

\* A Component consists of :

a) Presentation (done by html) (.html)

b) Styles (done by CSS) (.css)

c) Logic (done by Javascript/ JSX/ TypeScript) (.js|.jsx)

\* Reusability

\* Better Code Organisation

\* Follows Component-Based Architecture

Types of Components

o Functional Component (Recommended)

Syntax:

```
function ComponentName() {
  return (
    <div>
      JSX UI
    </div>
  );
}

export default ComponentName;
```

(Ex): function Welcome() {
 return (
 <div>
 <h1> Welcome to React </h1>
 </div>
 );
}

export default Welcome;

import Welcome from './welcome';

function APP() {

return (

<div>

<Welcome/>

</div> ); }

Ex: Header, footer, Main, Nav, Section file can be built by using components.

### 2) Class Components (Legacy or Not Recommended)

Syntax:

```
import React, { Component } from 'react';

class ComponentName extends Component {
  render() {
    return (
      <div>
        JSX UI
      </div>
    );
  }
}
```

Ex:

```
import React, { Component } from 'react';

class welcome extends Component {
  render() {
    return (
      <div>
        <h1>Welcome to React</h1>
      </div>
    );
  }
}
```

Note:

\* React applications are built using the Javascript ES6 module system, where each component file acts a module that exports and imports code.

### Ways to Add CSS in JSX (React):

#### 1) External CSS File

\* You have to create a file with .css extension and import into your component.

Ex: .title {

```
color: blue;
font-size: 24px;
```

}

3) Inline CSS (style Attribute)  
 In this, JSX uses Javascript objects for styles.  
Ex:  
 function App() {
 return (
 <h1 style={{ color: "red", fontSize: "24px" }}> </h1>
 );
 }
 }

Note:  
 the below rules applies to inline styles only.  
 1) CSS property name must follows CamelCase  
 Ex: backgroundColour, fontsize, textAlign, marginTop.  
 2) CSS Module (Recommended)  
 CSS module are way to write CSS that is scoped locally to a component.  
 \* Creates a module file  
 • File name must be ends with .module.css

Ex: .button {

```

background-color: blue;
color: white;
padding: 10px;
}
  
```

```

3
import styles from './button.module.css';

function Button() {
  return (
    <button className={styles.button}>Click Me</button>
  );
}

export default Button;

```

#### \* If multiple classes

- card s
 

```

border: 1px solid gray
}
      
```
- highlight s
 

```

background-color: yellow;
}
      
```

`<div className={ `${ styles.card } ${ styles.highlight }` }>`

`</div>`

- \* Props in React:
- \* props are immutable (cannot be changed)
- \* Props (properties) are way to pass data from a parent component

- \* to a child component
- \* props can only read the data, not change it
- \* props like arguments to a function

#### \* EX: Child Component

```

export function StudentCard(props) {
  return (
    <div>
      <p>ID: ${props.id}</p>
      <p>Name: ${props.name}</p>
      <p>Course: ${props.marks}</p>
    </div>
  );
}

```

#### \* props with destructuring (Recommended)

```

export function StudentCard({id, name, marks}) {
  return (
    <div>
      <p>ID: ${id}</p>
      <p>Name: ${name}</p>
      <p>Course: ${marks}</p>
    </div>
  );
}

```

## Parent Component

< StudentCard

```
id = 51
name = "Thru"
marks = 85
```

1 >

Note:

We cannot modify props "inside a child component"  
(or) props can only Read, not changed.

EX:- props.name = "John";

name = "Thru";

id = 51;

marks = 85

Props in class Components:

{ this.props.name } = ~~• this.props.name~~; (or) { this.props.propNames }

Syntax:

Fragments in React:

\* Fragments let you group multiple elements without adding an extra node to the DOM, unlike div.

\* In React a component must return one parent element. Fragments solve this.

\* Cannot add styles or props to fragments (except key)

Syntax: \* we can pass the key

(or)

< >

< React.Fragment >

< h1 > Hello < /h1 >

< p > Welcome to React < /p >

< p > Welcome to React < /p >

< /React.Fragment >

< />

\* cannot accept key

< >  
< /> } → Fragments

Rule:

\* A Component must Return a single parent

\* Key are required when using < React.Fragment > < /React.Fragment > not < > < />

\* React does not create extra node

..... styles, className, props) etc.

## What is One-way Binding?

- \* One-way Binding means data flows in only one direction
- \* One-way Binding means data flows from parent to child

## Hooks in React

- \* Hooks are nothing but a new features, also called as "in-built functions"
- \* Hooks are introduced in the React V16.8 version
- \* Hooks are backward-compatible with old features of ReactJS which means it does not contain any breaking changes

### Basic Hooks

- 1) useState()
- 2) useEffect()
- 3) useContext()

### Advanced Hooks

- 4) useRef()
- 5) useReducer()
- 6) useSelector()
- 7) useDispatch()
- 8) useMemo()
- 9) useCallback()
- 10) useId()

### Hooks for React Routing

- 11) useHistory()
- 12) useParams()
- 13) useLocation()
- 14) useRouteMatch()

### FAQs

- 1) Why or When we use Hooks?
- \* we will maintain state in Functional Component
- \* fetch and consume data (3rd party APIs) from Back-end
- \* to implement React Routing

### Rules of Hooks:

- 1) Call hooks methods ~~at top level~~ from react functions only
- 2) Only call hooks at the top level i.e; don't call hooks inside loops, nested functions, class components, conditions etc

## 1) useState()

\* useState() is a React hook that lets you add state to functional components

\* It is used for to store data and update dynamically in functional component  
 \* when state changes, the component re-renders and update data in us.

const [state, setState] = useState(initialValue);

↓              ↓  
 current value    function to update  
 of state        state

↓  
 starting value (number, array, string, object,  
 boolean etc)

\* useState() - is equal to the this.state and this.setState() method in class component.

Ex:- import {React, useState} from 'react';

```
function useprofile() {
  const [count, setCount] = useState(0);
  const [user, setUser] = useState(
    { "name": "thiagu",
      age: 18,
    });
}
```

usecase

- 1) Form handling
- 2) Button clicks
- 3) User interaction
- 4) Counter
- 5) Toggle

return (

<div>

<p>Count : {count}</p>

<button onClick={()=> setCount(count+1)}> Increase count </button>

<p> Name: {user.name}</p>

<p> Age: {user.age}</p>

<button onClick={()=> setUser({...user, age: user.age+1})}>

</button> </div>

> ; >

## 2) useEffect()

- \* useEffect() is one of the hook method.
- \* It is used for → perform the side Effects <sup>(API)</sup> in your components
- \* It is use for a performs the side Effects after a component renders-F(A))

Q When we use useEffect() ?

- \* Fetching the data from API (3<sup>rd</sup> parties)
- \* Updating the dom manually
- \* Cleanup operations ( setting timers (setTimeout, setInterval)).

Syntax:

```
useEffect( () => {
    // Code to Run (side Effect)
}, [dependencies]);
```

1<sup>st</sup> argument:

- this is a function that runs your side effect
- fetching data from API
- update dom manually
- cleanup operations.

2<sup>nd</sup> argument

optional array called dependency array.

- Controls when the Effect function [first argument] should run or be run.

how it will work ?

- In Class Components, we have lifecycle methods like ComponentDidMount, ComponentDidUpdate, ComponentWillMount.
- In functional Components, useEffect() can handle all three methods, depending on the second argument (dependency array).

\* `useEffect()` will depend on second argument.

### 1) Component Did Mount ()

\* Runs only once component is mounted.

\* When we use it:

- Fetching data from APIs
- Settimers

Syntax:

```
useEffect( () => { }, [ ])
```

\* When we write like above syntax, then Runs only once component is mounted.

"import {useState, useEffect} from 'react';"

Ex:- function API () {

```
const [users, setUsers] = useState([]);
```

```
useEffect( () => {
```

```
    fetch("https://jsonplaceholder.typicode.com/users")
```

- then ((response) =>

```
        response.json()
```

```
    }
```

- then ([json] =>

```
        {
```

```
            setUsers(json)
```

```
        };
```

```
    };
```

```
}, [ ]);
```

```
return (
```

```
    <>
```

```
        { users.map((us) => {
            </i> { us.user } </i>
        )) }
    ); } </>
```

## ② ComponentDidUpdate()

- \* It is a life cycle method in class components
  - \* It runs after component updates (i.e., after state or props change)
  - \* Runs when state or props change
  - \* When the props or states change the useEffect will call
  - \* Simply, when values change, runs logic (re-render)
- Syntax:

```
useEffect( () => {
    // [dep1, dep2 etc];
}
```

### useEffect:

- \* Reacting to state/change props change
- \* When we want to run code after something changes.

Ex:- const [count, setCount] = useState(0);

```
useEffect( () => {
    console.log(`Count: ${count}`);
}
```

```
    // [count];
```

```
return (
```

```
<div>
```

```
<h2> Count: ${count} </h2>
```

```
<button onClick={ () => setCount(count+1) }> Increase count
```

```
</button>
```

```
</div>
```

```
);
```

```
}
```

3) ComponentWillUnmount():

\* Runs after every Render.

\* It is used to perform cleanup operations such as timers, event listeners.

Syntax:

```
useEffect( c ) => {
    // cleanup function
    return () => {
        //
    };
};
```

(1) If second argument will have a property or state in array than depend upon that particular value update the useEffect will call.

(2) If second argument will have empty empty array only one time will appear after the component render.

(3) If second argument will empty every time component will render automatically this useEffect will call.

Ex: Cleanup operation

Ex:

```
const [count, setCount] = useState(0);
```

```
useEffect( c ) => {
```

```
let timer = setInterval( c ) => {
    setCount( count + 1 )
};
```

2,000)

```
return () => clearInterval(timer)
```

```
}, [] );
```

FAQ

1) What happens if you don't pass a second argument (no dependency array)?

- Effect runs after every Render

- This means it executes infinite times.

2) What happens if you pass an empty array [] as second argument?

Effect runs only once, after the first render

3) What happens if you pass dynamic data (state or props) in the dependency array?

The effect runs after first render.

Then it runs again whenever any value in the dependency array changes.

## Custom Hook in React:

\* A custom hook is a javascript function.

- \* Custom hook starts with 'use'
- \* Use built-in hooks (useState, useEffect etc)
- \* It allows you to reuse logic, not UI
- \* Custom hook = logic only.

### Benefits of Custom hook:

- 1) Code Reusability - write once, use anywhere
- 2) Cleaner Components - UI + Logic Separation
- 3) Better Readability
- 4) Easy to maintain and test

### When should you create a custom hook?

- \* Same logic is used in multiple components
- \* Component has complex logic then we will use custom hooks
- \* Logic is related to:

- 1) API Calling
- 2) Form handling
- 3) Authentication
- 4) Timers, Counters etc

### Syntax:

```
function UseCustomerHook()
```

{

// useState, useEffect. etc

return something;

}

|                  |                                              |
|------------------|----------------------------------------------|
| Normal Component | Custom hook<br>Name starts with <u>use</u> ) |
| ✓ Return JSX     | ✓ Responsible for UI<br>Rendering            |

✓ Responsible for UI  
Rendering

- 3) Handles HTML elements, lists, tables, dropdown etc.
- 4) UI
- 5) Face

✓ Responsible for only logic, not UI

- 3) Returns data + functions
- 4) Reusable across multiple components
- 5) Logic
- 6) Brain ✓

### Rules of custom hooks:

- 1) Name starts with use

Ex: useFetch, useCounter, useAuth

- 2) only call hooks:

- At top level
- Inside React functions or custom hook.

- 3) do not call hooks inside loops, or conditions

What is OneWay and TwoWay Binding?

OneWay-Binding:

\* Data flows in one direction (uni-directional)

\* data <sup>flow</sup> from JS → HTML

\* React follows One-way Binding by default.

Ex: const [name, setName] = useState("Ramu");

return (

<div>

<h2> {name} </h2>

<button onClick={() => {}}

setName("Shira");

}>

change </button>

</div>

)

Two-way Binding (using onChange) ✓

\* Data flows in both directions

\* Data flows from JS  $\leftrightarrow$  HTML which means JS to HTML and from

HTML back to JS. Using

\* Two-way achieved by combining value and onChange handlers

Ex: const [name, setName] = useState("Ramu");

return (

<div>

<h2> {name} </h2>

prop(s) variable

(or) connecting state with the UI, when data

\* Data Binding means Connecting data (state) with the UI, automatically updates changes, the UI automatically updates

\* states, props, variables, expressions inside JSX, must use in this {} }

```

<input type="text" value={name} onChange={()=>
    setName(e.target.value)} />

</div>
);

```

What is Prop Drilling?

The concept of passing data from parent component to all its child components that means at every level components - ~~using~~ using props. This called prop drilling.

Ex: App



What are the issues or problems of props drilling?

\* Passing data through multiple components is not good way.

Ex: Suppose you are having 200 components in your application

Parent - 1

Child - 2

Child - 3

Child - 199

Child - 200

- Actually I need to pass data from 1st component to 200 component

- but we cannot pass data directly from 1st component to 200 component

Using props - so we have to pass data through every level components

- Even though Components 2 to 199 Components did not need the state, they had to pass data state to bottom Components.

\* If anyone of Components is not working then data will not pass to the destination/target Component. →  
props drilling Issue

\* If we are passing data from 1<sup>st</sup> component to 200 component, in the middle if any one of component not working then entire communication will fail.  
\* Un-necessary data has to pass to every level of Components.  
to solve those problems / Issues we have to use

- 01) `useContext()` (or) Context API
- 02) Redux architecture
- 03) Component Composition

### What is Context?

\* Context is a one of the Top level API in React JS

\* Using Context - we can share data between multiple components without having any relationship between them

\* that means Context API will provide global communication between components - to pass the data from one component any component directly without using props.

### FAQ)

Q) How can you pass directly data between multiple components using Context API? or How we use Context...?

There are two main steps to use React context into React application:

01) Setup a Context provider and define data which you want to pass to other Components.

02) Use a Context consumer whenever you need the data from store.

## Q1 How to Create Context in your React Application?

- Using `React.createContext()`

```
const Context = React.createContext();
```

- `React.createContext()` will give

`Context.Provider` → it will create Global store pass the data

`Context.Consumer` → It is used consume data of a provider

## Q2 What is Context Provider?

- Context Provider is nothing but a Global store that will give data

→ to any component

- Child Components need to be include in Provider Component

```
<Context.Provider value={name}>
  <Component1/> value → {name}
  </Context.Provider>
```

Ex: `<Context.Provider value='Hello'>`  
~~<Component1/>~~  
`</Context.Provider>`

## Q3 What is Context.Consumer?

`Context.Consumer` → It is used consume data of provider

```
<Context.Consumer>
```

{value}

```
</Context.Consumer>
```

### Syntax

```
<myContext.Provider>
```

`value={user}`

```
</myContext.Provider>
```

→ Provider

Recommended  
(Modern way)

✓  
const user = useContext(MyContext);

(or)

`<MyContext.Consumer>`

`</MyContext.Consumer>`

Consumer

## Q) What is useContext() ?

- \* useContext() is a hook method
- \* Return context value
- \* Which is used to consume data in functional component
- \* It avoids props drilling.

Syntax:

```
const data = useContext(ContextProvider);
```

## What is Context API ?

- \* A React feature
- \* It is the mechanism to create and provide the global data
- \* It includes:

- 1) createContext()
- 2) Provider
- 3) Consumer

- \* (string, array of objects, objects of arrays, number, boolean, null / undefined) data we can pass in the Context Provider.

Note: When the provider value changes, all components consuming that context using useContext will re-render to updated value.

Ex: import {useContext} from 'react';

```
const userContext = React.createContext();
```

export function Parent() {

```
const user = {
```

```
  name: "Hirau";
```

```
  age: 20;
```

return [

~~<div>~~

~~<p> name: {name}</p>~~

~~<p> age: {age}</p>~~

<UserContext.Provider value={user}>

~~<Child/>~~

</UserContext.Provider>

```
}
```

```
function Child() {
```

```
const user = useContext(UserContext);
```

```
return [
```

```
<div>
```

```
<p> name: {name}</p>
```

```
<p> age: {age}</p>
```

```
</div>
```

```
];
```

}

```
,];
```

5) useRef(): → stores mutable values

\* useRef() is a hook method

\* Which is used to create a reference to the DOM elements in the functional components

\* Performance increased by accessing DOM elements directly

\* It can be used to access a DOM element directly

(09)

\* Which is mainly used to persist values between renders

\* It can be used to store a mutable value that does not cause a re-render when updated

\* useRef returns mutable object whose current property

Syntax: can be update without re-rendering

→ same object is used across all renders.

const refObject = useRef(initialValue); → It is an object

{current: initialValue}

Const inputValue = useRef(""); → Ref Object

Const count = useRef(0); → Ref Obj

Count.current → to present value store

Count.current = "It holds present value";

FAQ) ~~num~~ current.name = "three"; → updating the data

i) Where / when do we useRef()?

Re-Rendering will not happen

accessing DOM element

OnFocus - input field

Render () - counting

Stop watch

Q) What is the difference between useState() vs useRef()?

\* useRef() does not cause re-renders when state is updated

\* useState() cause re-renders when the state is updated

Ex:-   
 Const [count, setCount] = useState(0);  
 useState : return the current state value.  
 It has a setter method that updates the state and It is  
 Causes components to Re-Renders

Const count = useRef();

useRef() : returns an Object,

It is used to access DOM elements directly but it does not  
 cause components to Re-Renders

Ex:-

```
import React, {useRef} from "react";
function InputFocus() {
  const inputValue = useRef("");
  function focusInput() {
    inputValue.current.focus();
  }
  return (
    <div>
      <input ref={inputValue} type="text" placeholder="Type here..."/>
      <button onClick={() => focusInput}>Focus Input</button>
    </div>
  );
}
export default InputFocus;
```

## 6) `useReducer()`

\* `useReducer` is a hook used to manage complex logic state logic in functional components.

\* Returns an array.

Faq)

1) What is a Reducer<sup>in</sup>(Redux) / `useReducer`) ?

\* A reducer is a pure function that is used to update the state of an application

\* A reducer :

Accepts previous state and an action

Calculates the next time

Returns a new state Object

Does not mutate the original state

i.e;  $(\text{previousstate}, \text{action}) \Rightarrow \text{newstate}$

Previousstate State (initialstate)

Initialstate

Ex: const initialState = { count: 0 }

Reducer Function syntax:

function reducer ( state = initialState, action )

{

    return newState;

}

Q) What is an Action?

A) An action is a plain Javascript object that describes what happened.

Structure of Action:

{  
  type: "INCREMENT",  
  payload:  
    name: "thru",  
    age: 20,  
    Count: 1  
}

Q) What is dispatch?

A) dispatch is a function used to send an action to the user.

Ex: <button onClick={() => dispatch('increment')}> add </button>

<button onClick={() => dispatch({ type: 'increase' })}> add </button>

<button onClick={() => dispatch(actionCreator)}> add </button>

Q) What is useReducer()?

A) useReducer is a React hook

\* Similar to useState() but it is used for complex state management

\* follows Redux-style pattern (State + Action → Reducer → newState)

\* useReducer() is a React hook that manages states by sending actions to reducer, which return a new state

Q) Why we use useReducer()?

\* state logic is complex

\* Complex logic

Q) How does useReducer() work?

Component loads with initial state

dispatch(action) is called

React calls reducer(initial state, action)

Reducer returns new state

Component re-renders with update

Use Reducer Syntax:

```
const [ state, dispatch ] = useReducer ( reducer, initialState );
```

reducer → function to update

initialState → starting state

state → current state

dispatch → function to send actions

Work flow:

Initial State → UI shows data



User clicks button



dispatch (action)



reducer (state, action)



Switch (action-type) ↳ called React automatically



new State returned



React updated state



UI re-renders with new data

\* When reducer returns a new state object, React triggers a re-render

What is the difference between useState and useReducer?

### useState

- 1) You update directly (direct approach)
- 2) Logic inside the component

```
onClick={() => setCount(count + 1)}
```

- 3) Good for simple state, single values

### useReducer

- 1) You send an action (action-based approach)
- 2) dispatch (type: "INCREMENT")

- 2) Logic is outside the component (in reducer)

```
function reducer(state, action) {
  if (action.type === "INCREMENT") {
    return state + 1;
  }
}
```

Good for complex state or large components, multiple related values, conditional updates or when multiple actions affects the state.

### 7) useMemo()

What is useMemo?

- \* useMemo is a React hook that memorizes (caches) the result of a calculation and re-recomputes (or) re-renders it only when its dependencies change.

\* performance optimisation

- \* Why we use useMemo?

\* React re-renders a component when state or props change.

- \* On every render, all calculations run again

\* Heavy calculations can slow down the app

- \* by using, useMemo avoids necessary re-calculations.

Syntax:

```
const memoizedValue = useMemo(() => {
    return calculation;
}, [dependency]);
```

Ex:

```
import { useMemo } from 'react';

function App({num}) {
    const square = useMemo(() => {
        return num * num;
    }, [num]);
    return (
        <h1> {square} </h1>
    );
}
```

React calculates  $num * num$

Stores (memorizes) the result

React Re-uses the stored value on re-render if num does not change

Note:

`useMemo()` does not stop re-render

It stops unnecessary re-calculations

No array → Recalculates on every Render

Empty [] → Calculate only once

[state / props] → Recalculates when data changes.

What is memo() in React?

\* React.memo is not a hook method  
\* memo (written as React.memo)

\* It is a High Order Component (HOC) used to prevent unnecessary re-renders of a component

\* It will do two things:

- 1) It will stop unnecessary re-rendering of a functional component if its props or state do not change.
- 2) It will improve the performance of a functional component.

Syntax:

```
const MemoComponent = React.memo(Component);
```

```
Ex: import React, { useState } from 'react';
```

```
const Child = React.memo(({ value }) =>
```

```
  console.log(`child rendered`);
```

```
  return <h2> Value: {value} </h2>
```

```
});
```

```
function Parent() =>
```

```
  const [count, setCount] = useState(0);
```

```
  return (
```

```
    <>
    <Child value={10} />
```

```
        <button> onclick = {() => SetCount (count + 1)} > Parent Re-renders if count > buttons  
    </>  
};  
}
```

8) use callback()

- 8) `useCallback()`
    - \* `useCallback()` is a React hook that memorizes (caches) a function so that the same function reference is re-used between renders.
    - \* To prevent unnecessary re-renders of child components.
    - \* It returns a memoized version of a component that only changes when its dependencies change.

### Syntax:

```
Syntax:  
const memoizedFn = useCallback(() =>  
    {  
        // logic  
    }, [dependencies]);
```

`afc| afce| mafce` (vscode snippets in functional Components)  
export with export

- 1) `rfce + Enter` → React Functional Component with export  
2) `rfc + Enter` → React functional Component (no export, default).  
3) `rafc + Enter` → React Arrow functional Component with export

## Conditional Rendering or Ternary Operator:

\* It is a javascript operation.

Syntax:

Condition ? ExpressionIfTrue : expressionIfFalse

Ex:

{ isLogin ? <h1> Welcome back! </h1> : <h1> Please Login </h1> }

Nested ternary for multiple condition:

```

if status == "loading"
    ? "loading...""
    : status == "success"
        ? "success!"
        : "Error"
    }
}

```

Logical AND ( && )

\* Only if condition is true

Syntax:

Condition && <JSX-to-render-if true/> (or) Condition %&% expression.

- if condition is true, then React renders the JSX

- if condition is false, then React renders nothing (skips it)

Ex:- { isAdmin = {true} } && <button> Delete user </button> }

## Events in React

An Event is an action triggered by the user or browser  
 Ex: Clicking a button, Typing in input field, submit a form etc.

### Event Rules in React:

1) Event names are written in camelCase

Ex: onChange, onSubmit, onClick etc

2) pass the function reference, not functional call

Ex: `onClick={handleClick() }` X  
`onClick={handleClick}`  ✓

Event object has properties like:

`event.target` → element that triggered the event

`event.preventDefault()` → stops the behaviour

`event.type` → type of event

### Synthetic event

React event object  
 (e) passed into your functions

event listener synthetic event

`<button onClick={()=>`

↑  
`{handleClick}` ↑  
 ↓  
 callback  
 function

Ex:  
`function handleClick()`  
`{`  
 `alert("Button clicked!");`  
`}`  
`return (`  
`)>`  
`<button> onClick={handleClick} click here </button>`  
`} </>`

(or)

function handleClick

`<button onClick={()=>`

`alert("Button clicked!");`

`}`

click here

`</button>`

|                                |                    |                |                     |
|--------------------------------|--------------------|----------------|---------------------|
| 1) Mouse Events                | 2) Keyboard Events | 3) Form Events | 4) Clipboard Events |
| a) onclick                     | a) onkey down      | a) onchange    | a) oncopy           |
| b) ondoubleclick               | b) onkey up        | b) onsubmit    | b) onpaste          |
| c) onmouseenter                | c) onkeyup         | c) onfocus     | c) oncut            |
| d) onmouseleave                |                    | d) oninput     |                     |
| e) onmousedown                 |                    | e) onblur      |                     |
| f) onmouseup                   |                    |                |                     |
| g) onmousemove                 |                    |                |                     |
| h) oncontextmenu (Right-click) |                    |                |                     |

### Dynamic Expression:

- dynamic expression is any javascript expression written inside {} in jsx.
- React evaluates it at render time and shows the results in the UI.

#### Syntax:

{ expression }

Ex: let name = "Thirumalai";  
<hr> Hey Hello {name} </hr>

### Template Literals:

- Template Literals are Javascript strings that allow interpolation (inserting variables or expressions inside a string) by using backticks and {}{}.

#### Syntax:

' string {{expression}} '

Ex: let name = 'Thiru';  
let age = 22;

let msg = ' My name is {{name}} and I am {{age}} years old ';

## Map

- \* Map() is a Javascript array method
- \* It loops through an array and returns new array
- \* It is commonly used to render list of elements in React
- Ex, lists (<ol>, <ul>) , dropdowns, tables, cards/ components from API data

Syntax:

```
Ex: const names = [{id:1, name:"thiru"},  
                   {id:2, name:"shiva"}]  
  
array.map((item) =>  
  <JSXElement/>  
)  
}  
  
return(<ul>  
  {names.map(users =>  
    <li key={users.id}>  
      {users.name}  
    </li>  
  )}  
</ul>)
```

### Faq

What is a key !

when

\* Key is a special attribute used <sup>when</sup> rendering lists using map().

\* Key is used for uniquely identify list of Items

\* Key helps React identify which items in a list have changed, been added, or removed, so it can re-render efficiently and correctly.

## Lists:

\* Lists are used to display multiple similar items

\* <ol>/<ul> , dropdowns (<select>), tables

\* Javascript arrays + map() to create lists.

## Arrays:

\* Array stores multiple values in a single variable

\* Array + map() = Dynamic UI.

\* Filter / sort / search data etc.

[ { } ] → Array of Objects

{ [ ] } → Objects of Array

## Syntax:

```
const users = [
    { id: 1, name: "Thivu", age: 22 },
    { id: 2, name: "Anusha", age: 22 },
    { id: 3, name: "Sreedevi", age: 23 }
]
```

## Note:

(\*) Dot Notation is used to access object properties directly using the property name.

Ex: Object.PropertyName

Ex: users.name, users.age

## Objects

\* It stores data in key-value format

## Syntax:

```
const users = [
```

```
    { name: "Thivu",
      age: 22,
      city: "hyd"
    },
    { name: "Shiva",
      age: 23,
      city: "Bang"
    }
```

| accessing object value ( . Notation )

<p> {users.name} </p>

<p> {users.age} </p>

<p> {users.city} </p>

}

]

## React Router

- \* React Router is used for Client side Routing in React SPA
- \* What is the React Router Version?

V6 (latest version)

### Q) What is Routing in React

- \* Routing = Mapping URL → Component
- \* When the URL changes, different component Renders

Ex:-

/home → Home Component

/about → About Component

### 3) Difference between Router and Routing

#### Router

- \* Router = tool / mechanism
- \* Router is a component or library that handles navigation in an application
- \* BrowserRouter is the Router
- \* It listens the URL changes

#### Routing

- \* Routing = process / Configuration
- \* Routing is the process of defining paths and mapping them to components
- \* Router is the Routing

E

Ex: <BrowserRouter> { \* / Router \*}  
 = <Router> { \* / Routing starts \* / }

<Route path = "/about" element = { <About /> } />

</Routes>

</BrowserRouter>

Advantages:

\* No page Reload (SPA Behaviour)

\* Clean and understand URLs

\* Dynamic Routing Support

\* Programmatic Navigation

\* Nested Routing Support

\* 404 and error handling

Difference between SPA and MultiPage Applications.

SPA (Single Page Applications)

MPA (Multi Pages Applications)

\* Only one HTML page (index.html)

\* Each URL = new HTML page

\* Javascript Controls navigation

\* Backend Control Navigation

\* No page Reload

+ full page Reload

Ex: React, Angular, Vue

Ex: JSP, PHP, Thymeleaf.

How we install React Router?

npm install react-router-dom

1) BrowserRouter

\* BrowserRouter is the main router Container

\* It uses HTML5 History API to keep the UI in sync with the URL

\* All Routing Components in-side it.

\* Enables client-side Routing

\* Use only once, at the top of app

Syntax:

```
import { BrowserRouter } from 'react-router-dom';
<BrowserRouter>
</BrowserRouter>
```

## 2) Router:

- \* Router is a Container for all Route Components
- \* It renders only matching Route
- \* Groups Multiple Routes
- \* Decides which Component to render based on URL

Syntax:

```
import { Router } from "react-router-dom";
```

```
<Router>
  <Route Components ...>
</Router>
```

## 3) Route:

- \* Route maps a URL path to a Component
- \* When the URL matches the path, the component renders.

Syntax:

```
import { Route } from "react-router-dom";
```

```
<Route path="/home" element={<Home>} />
```

Basic Syntax

```
<BrowserRouter>
  <Router>
    <Route>
      <Route />
    </Route>
  </Router>
</BrowserRouter>
```

Ex:

```

import React from "react";
import { BrowserRouter, Route, Route } from "react-router-dom";

function Home() {
  return (
    <h2> Home Page </h2>
  );
}

function About() {
  return (
    <h2> About Page </h2>
  );
}

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/home" element={<Home />} />
        <Route path="/About" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;

```

Attributes:

- path → defines the URL path
- element → specifies which component to render
- index → \* default page inside a parent Route
  - \* NO path
  - \* only one index routes can a parent have
  - \* A default child route that renders when parent route matches

## Navigation: static :

Navigation means moving from page/screen to another page/screen in an application.

### 1) Link (Navigation)

- \* <Link> is a React Router Component used for Navigation between pages without page Reload.
- (Q9)
- \* <Link> is used to Navigate from one route to another route in a React app without Refreshing the page

#### Syntax:

```
import { Link } from "react-router-dom";
<Link to="/about" > About </Link>
```

\* NO Reload

\* Fast Navigation

\* SPA Behaviour

Difference between <Link> and <a> (anchor) tag ?

<a> (anchor) tag

\* Reloads entire the page

\* No SPA Behaviour

\* Slower

\* Used for External links

\* attribute: href

<Link>

\* No page Reload

\* SPA Behaviour

\* Faster

\* Used internal React routes

\* attribute: to

→ <a> tag is used to navigate from one page to another page by Reloading page

What is to attribute?

\* "to" attribute is a prop of React Router's `<Link>` and `<NavLink>` that tells where to navigate.

\* to specifies the destination path(URL) for navigation in React Router.

Syntax:

```
<Link to="/search"> Search </Link>
```

Object Form (-Advanced)

`<Link`

`to = {`

`pathname : "/search",`

`Search : "q=search",`

`hash : "#top",`

`}`

`> Search </Link>`

~~syntactic~~  $\Rightarrow$  NavLink (navigation + active styling)

\* `<NavLink>` is a special version of `<Link>` in React Router that is used for navigation with active state.

\* `<NavLink>` is used to navigate b/w routes and automatically detect whether the link is active or not.

Syntax:

`<NavLink`

`to = "/home"`

`style = { ( isActive ) => (`

`color: isActive ? "Green": "Red",`

`) ) }`

`> Home`

`</NavLink>`

## Programmatic Navigation: (03) Dynamic Navigation

- \* Navigate is used to move from one route to another using Javascript Logic instead of clicking event
- \* `useNavigate()` is a hook that allows navigation between routes<sup>^</sup> using Javascript logic instead of clicking links
- \* `useNavigate()` is a React Router hook, which gives `navigate()` function, which is used for Programmatic Routing

Syntax:

```
import { useNavigate } from "react-router-dom";
const navigate = useNavigate();
```

Ex:

```
function Home() {
  const navigate = useNavigate();
  return (
    <button onclick={() => navigate("/profile")}>
      Go to profile
    </button>
  );
}
```

Navigation with delta values :

- \* Negative delta → Go back
- \* ~~Positive~~ positive delta → Go forward

Syntax:

```
navigate (-1); // back 1 page
navigate (+1); // forward 1 page
navigate (-2); // back 2 pages
navigate (0); // page Refresh
```

Navigation with params :

```
navigate(
  {
    pathname: "/user/10",
    search: "?role=admin",
    hash: "#!",
  }
);
```

```
navigate (" /user/10")
navigate (" /products ?category=mobile
          &page=2");
```

Nested Routes :

What is Nested Routes?

- \* A route inside another route, sharing a common layout.
- \* Nested routes are used when one page layout stays common, and only part of the page changes.

Ex: /dashboard → parent route  
 | dashboard | profile ↗ child routes  
 | dashboard | settings ↘

- \* To reuse layout
- \* To avoid repeating header/navbar

Syntax:

```

<BrowserRouter>
  <Routes>
    <Route path = "dashboard" element = {<Dashboard />}>
      <Route path = "profile" element = {<Profile />}>
      <Route path = "settings" element = {<Settings />}>
    </Route>
  </Routes>
</BrowserRouter>

function Dashboard() {
  return (
    <h2> Dashboard </h2>
    <Outlet/>
  );
}
  
```

↗ placeholder Component  
 <Outlet/> ↗ used in a parent Route  
 to display its nested child routes.

Rules in Nested Routes:

- 1) Parent Route must have <Outlet/>
- 2) Child <Route> must be inside Parent <Route>
- 3) Child path are RELATIVE (no /)

## Error Page handling in Routers:

Error handling in routing means showing proper UI when navigation fails, instead of a blank screen.

Ex: User enters the URL → Page does not found or Unauthorized access.

404 → Page not found,

401 → Unauthorized

403 → Forbidden

500 → Server error

## Syntax:

```
<Route path="*" element={<div><h2>404 - Page Not Found </h2></div>}>
```

< /Route >

\* path="\*" should be last, because the router checks routers top-to-bottom

### Note:

"\*" → It is a wildCard

"\*" matches Any URL that does not match other defined routes

"\*" matches Any URL that does not match → \* route handles it.

"\*" React Router does not find a match → \* route handles it.

## Route Params / path Params in React Router (v6)

What are Route Params / path Params? → variables inside the URL path

Route Params (Path Params) are dynamic values in the URL path.

\* Route Params / path params mean Params defined in route path

\* they allow us to load different data using the same Component

\* path params are dynamic values in the URL path used to identify specific data

Advantages: 5) Support RESTful Design

1) Dynamic Routing

2) Reduces code duplication

3) SEO Friendly and shareable URLs

4) Simplified the Data Fetching (Fetch data directly based on the URL)

## Static Path Params / Route Params

What are static path params

static path params are fixed path in the URL that don't change

Syntax:

```
<Route path="/about" element={<About/>} />
```

## Dynamic path Params / Route params / Dynamic segments:

Dynamic path Params / Route params / Dynamic segments: dynamic path in the URL that can change.

- Dynamic path Params are dynamic path in the URL that can change.
- Dynamic path params (or dynamic segments) are variables in the URL path (or)
- Dynamic path params (or dynamic segments) are variables in the URL path that can change depending on the data.
- By using ":" (semicolon) we can achieve the dynamic Route params.
- Always use ":" before param name

Syntax:

```
<Route path="/user/:id" element={<User/>} />
```

:id → dynamic segment

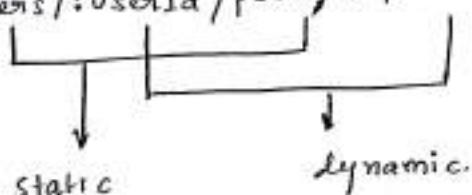
\* Multiple Dynamic Params

```
<Route path="/product/:category/:id" element={<Product/>} />
```

↓            ↓  
static path param   dynamic path param / dynamic Segment

\* Nested Dynamic Params

```
<Route path="/users/:userId/posts/:postId" element={<Post/>} />
```



## UseParams()

\* UseParams() is a React Router hook used to Read path (Route) Parameters from the current URL.

\* It allows a component to know which dynamic value is present in the URL.  
\* Works only with path params, not query param.  
Syntax:

```
const Params = useParams();
```

\* useParams() returns an object containing :

- key → param name
- value → URL value (always a string)

\* Values are always string

Ex:-  
const { id } = useParams();

```
{ id: "101" }
```

Ex:- <Route path="/student/:branch/:roll" element={<student/>} />

```
function student()
```

```
{  
  const { branch, roll } = useParams();
```

```
  return
```

```
(
```

```
<>
```

```
<h2> Student Branch : { branch } </h2>
```

```
<h2> Student RollNo : { roll } </h2>
```

```
</>
```

```
;
```

```
}
```

URL

/student/cse/23G3IA0536

Return Object :

```
{  
  branch: "cse",  
  roll: "23G3IA0536"  
}
```

## What is Optional Segment?

- \* An Optional Segment is a URL parameter that may or may not be present.
  - \* It is the part of the URL path, but not mandatory.
- Syntax:

: ParamName ?

< Route path = "/path/:ParamName?" element = {< component />} />

Ex: < Route path = "/user/:uname/:usage?" element = {< user />} />

URLs that work:

/user/thru

/user/thru/23

< Route path = "/user/:uname?/:usage?" element = {< user />} />

URLs that work:

/user/thru/21

/user/thru

/user

||

Search Params / Query Params in React Router:

What are Query params / Search Params?

\* Query Params / Search Params are key-value pairs added to the URL

after ? to send extra information to a page.

\* Query Params are optional (not mandatory) and shareable.

\* It is used to pass optional data in the URL, mainly for filtering,

sorting, searching and pagination etc.

Syntax:

/path?key=value

Ex: /users?role=admin

1 path ? key = value & key1 = value1      Ex: /user? name = shiva & age = 21 -

Advantages :

- 1) Shareable URLs
- 2) can handle multiple values
- 3) Best for filtering, sorting, searching

useSearchParams() :

- What is useSearchParams ?
- \* useSearchParams() is a React Router hook used to Read and update Query parameters
- \* (Search Params) in the URL
- \* It is used for only Query params
- \* Query params are part of the URL after ?
- \* Read Query params from the URL
- \* Update Query params without page Reload
- \* Maintain filter, search, pagination state in URL
- \* Make URL Shareable

Syntax:

```
const [ searchParams, setSearchParams ] = useSearchParams();
```

- \* It returns array with 2 values
- ↓                    ↓
- used to Read Query params      function used to update Query params

Two methods in useSearchParams:  
Important methods in useSearchParams:

1) get(key)

used to Read a single Query param value by Using key

2) getAll(key)

used to When same key appears multiple times

3) has(key):

Checks whether a query param exists.

4) delete(key): → to delete all keys with same key name.  
 (or) delete(key, name); → to delete particular key

Removes a query param completely.

5) forEach(callback)

loop through all query params

6) append(key, value)

\* Adds a new value without removing existing one

\* Allows multiple values for same key (Allows duplicate values)

7) set(key, value): (does not allow duplicate value).

\* Adds a new value with ignoring existing one (If it is already exists).

\* Does not allow multiple value for same key [set will not allow duplicate]

8) toString(): Converts query string params object back into a URL-encoded query string

Ex: /products?category=mobile&page=1

import {useSearchParams} from "react-router-dom";

function products() {

const [searchParams, setSearchParams] = useSearchParams();

const category = searchParams.get("category");

const changePage = () => {

setSearchParams(prev) => {

prev.set("page", 2);

return prev;

}

```

const addBrand = () => {
  setSearchParams(prev => {
    prev.append("brand", "samsung");
    return prev;
  });
}

const clearCategory = () => {
  setSearchParams(prev => {
    prev.delete("category");
    return prev;
  });
}

const hasCategory = searchParams.has("brand");

return (
  <div>
    <h2> products </h2>
    <p> Category: {category} </p>
    <button onClick={changePage} > changePage </button>
    <button onClick={addBrand} > add Brand </button>
    <button onClick={clearCategory} > delete category </button>
  </div>
)
export default products;

```

What is URLSearchParams?

URLSearchParams is a built-in JavaScript Web API that provides an easy way to Read, Manipulate, and Create Query parameters in the URL.

- \* React uses it internally for `useParams` `useSearchParams`.
- \* Represents the query string part of a URL (after ?)
- \* provides methods to read, add, update, delete, check parameters.
- `?category=mobile &page=2` → Query params
  - `URLSearchParams` helps to read or modify these params.

What is `useLocation`?

`useLocation` is React Router hook that is used to access current URL information (`pathname`, `search`, `hash`, `state`)

Ex `/user/21?name=thru #dev`

```
const location = useLocation();
console.log(location);
```

Output: {  
 pathname: "/user/21",  
 search: "?name=thru",  
 state: null,  
 hash: "#dev"

Difference b/w Path Params and Query Params?

Query Params

Path Params

1) Dynamic segment in the URL path

Key-value pairs in the URL after ?

2) `/user/:id` → `/user/21`

`/products?category=mobile`

3) usually Required  
 (identifies resource)

Optional (filter, sort, search)

4) Access params by using  
`useParams()`

Access params by using `useSearchParams()`

5) part of the URL path

after ?

6) @PathVariable

@RequestParam

provides extra info & filters about that resource

Ex: /user/21 → tells which user

How/which / optional info

7) Who/what

How many ways we can pass Query params?

1) Directly in URL (static)

you can manually write query parameters in the URL

Ex: /user?name=thiru&age=22

2) Using useNavigate or <Link> or <NavLink> (programmatic Navigation)

you can pass query params as part of navigation dynamically.

<Link to="/user?name=thiru"> Go to users </Link>

(Ex)

```
const navigate = useNavigate();
navigate("/user?name=thiru");
```

```
navigate({  
  pathname: "/user",  
  search: "?name=thiru"});
```

3) Using URLSearchParams / useSearchParams

What is search attribute?

\* Search attribute is part of URL that contains query parameters.

\* It starts with ? and includes key-value pairs separated by &.

Ex search: "? age=21"

## Lazy Loading :

Different ways to implement lazy loading.

- 1) With Conditional Rendering (or) Ternary Operator
- 2) with Routing

1) What is Lazy loading in React?

\* Lazy loading means loading a component only when it is actually needed, instead of loading everything at the initial page load (or) at once.

\* Faster Initial Page load

\* Better Performance / Improved Performance

\* Reduce Bundle Size

\* Makes app faster

\* Saves Bandwidth

\* Memory Efficiency (Less memory JS at once).

2) What is lazy () ?

lazy() is a React function, which is:

- used to delay Component code loading.
- Code downloads only when Component is Rendered
- lazy() expects One component to Render.

\* works only with default exports.

\* Must be used with <Suspense>

Syntax:

```
const ComponentName = lazy(() => import('./ComponentName'));
```

```
Ex: const Profile = lazy(() => import('./Profile'));
```

3) What is <Suspense> ?

- \* <Suspense> is a component of React.
- \* handle the waiting state
- \* Pauses the rendering until code loads/downloads.
- \* Waiting for something asynchronous.

4) What is fallback in React Suspense?

- \* fallback is a prop of suspense that displays temporary UI msg until the component finishes loading.
- \* Message to show while waiting / loading / downloading.

Syntax:

```
<Suspense fallback=<h3> Loading ... </h3>>
  <LazyComponent/>
</Suspense>
```

Ex: import {lazy, suspense} from "React";

```
const Profile = lazy(() => import("./Profile"));
```

function APP()

{

return

<>

{

show &lt;

```
<Suspense fallback=<h2> Loading ... </h2>>
```

```
</profile>
```

```
<suspense>
```

}

</>

}

Flow:

Condition true :

- React tries to render <Profile>
- lazy() triggers import("./Profile")
- Browser downloads profile.js.
- <Suspense fallback> is shown
- After download → Profile Render.
- Download → Render.

Note:

- 1) Lazy loading Control when the Component code is downloaded and lazy loading is not responsible for rendering
  - 2) React itself is responsible for rendering.
  - 3) Once a lazy-loaded Component's code is downloaded, it is cached and reused, <sup>js file stored in browser cache + memory</sup>  
It is not downloaded again
  - 4) Lazy loading works with only default exports.
  - 5) When the page refreshes then chunks download again
  - 6) When the user Re-Renders the page then it will reuse from memory.
- Note: → When the code is downloaded then only that code will be rendered/run

Lazy Loading in React Routing:

Route Components are loaded / downloaded only when the user visits that Route.

## Why lazy load routes?

## \* without lazy loading:

- All Routes → download at once → one big bundle.js

- slower initial load

## \* with lazy loading:

- Routes split into separate chunks

- . pastes first page load

- . improve performance and better for performance in large apps.

Syntax: → const Profile = lazy(() => import('./Profile'));

<const Home = lazy(() => import('./Home'));

<BrowserRouter>

<Suspense fallback={<div> Loading ... </div>}>

<Routes>

<Route path="/Profile" element={<Profile>} />

<Route path="/Home" element={<Home>} />

</Routes>

<Suspense>

</BrowserRouter>

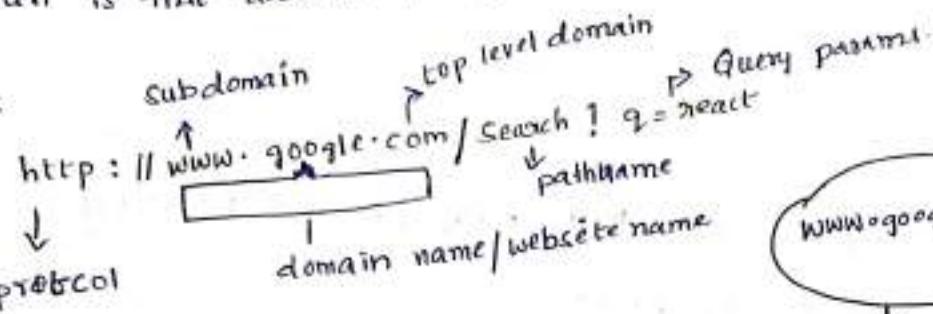
FAQ:

What is URL?

\* URL - Uniform Resource Locator

\* URL is the address of Resource on the internet  
(or)

\* URL is that address for particular website.

Ex: 

Full Qualified Domain Name (FQDN):

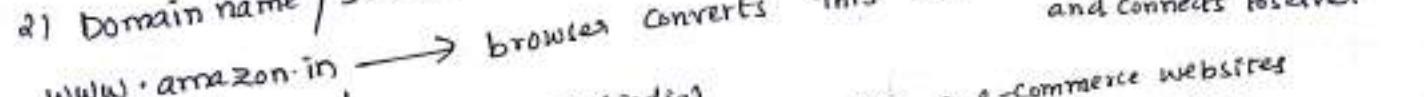
1) http (Protocol)

- HyperText Transfer Protocol

- http is used for communication between the client (browser) + server

- https → for security

2) Domain name / Server name :



www | .amazon | in  
Subdomain domain | top level domain (India)

• Which Server you are Requesting

.com → e-commerce websites  
.org → government sites

.in → India country

.us → USA country

.edu → education Institutions

3) Path

• Which Resource you want from the server

4) Query Parameters:

• Extra Information | filters | sorts | pagination

4) Port Number:

Port number that identifies which application should receive the requests

• default port: 80 (http)

default port: 443 (https) : amazon, meesho, flipkart, myntra, Any security site.

## API Integration:

Different ways to call API

1) fetch() method

2) axios library

3) jquery library

### 1] Fetch()

What is fetch() ? → to connect ~~and~~ backend and frontend

\* fetch() is a method provided by Browser.

\* fetch() is a built-in javascript API used to communicate with a

server (Backend) using HTTP methods.

\* used for call backend api calls in fetch

\* used for api integration

Syntax:

```
fetch(url, [options])
      ↓      ↓
    url       method, headers, body
```

How fetch() works ?

① Browser sends Request to Server

↓

Server responds (JSON/text)

↓

fetch() return promise

↓

Convert response using .json()

↓

use the data.

Ex: 1:

```

fetch ("https://jsonplaceholder.typicode.com/users")
  .then (response => response.json())
  .then (data => {
    console.log(data);
  })
  .catch (error) {
    console.log("Error", error);
  }
;

```

Ex: 2:

```

async function addData () {
  try {
    // Wait for the response before continuing
    const response = await fetch("https://fakestoreapi.com/products",
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          title: "new product",
          name: "thirumalai",
          price: 10.11,
          category: "men's clothing",
        });
      });
    if (!response.ok) {
      throw new Error(`Failed to post data`);
    }
  }
}

```

*Annotations:*

- A callout arrow points from the line `"Content-Type": "application/json"` to the text: "tells to the server what type of data you sending".
- A callout arrow points from the line `body: JSON.stringify({ ... })` to the text: "Converts Javascript objects into JSON".

Wait for Response body to be parsed into JSON before continuing.

```
const data = await response.json();
console.log(data);
```

→ Converts JSON (strings) from server into  
JavaScript object

catch (error) {  
 console.log(error);  
}  
}

JavaScript object is the actual  
usable data

Http methods → What actions the client (browser) wants perform on the server.

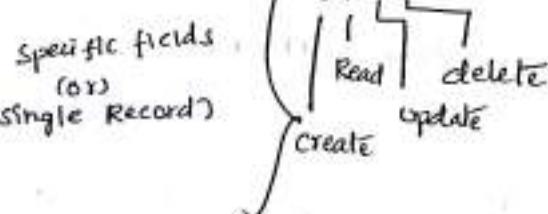
1) GET → Read Data

2) POST → Create Data

3) PUT → update full data

4) PATCH → update partial data (single Record)

5) DELETE → Remove / Delete Data



Http status code: → A numeric code returned by a server indicating

Request result for frontend

\* For communication, error handling, debugging.

2xx → Success  
Request processed successfully

1) 200 → OK - GET, PUT

2) 201 → Created → POST

3) 204 → No Content → DELETE

3xx → Redirection

Client must take another action

301 → moved permanently

302 → Found

304 → Not modified

4xx → Client Errors

Problem from client ~~server~~ side

400 → Bad Request

401 → Unauthorised

403 → forbidden

404 → Not found

5xx → Server Errors

Problem from server side

500 → Internal Server Error

502 → Bad Gateway

503 → Service unavailable

## Q) What is .json()

- .json() : Converts JSON (Raw Response) from the server into a Javascript Object
- Converts JSON → Javascript Object

## A) What is headers?

- Headers carry extra information about the Request or Response.

### 1) Request

headers: {

"Content-Type": "application/json" } → It tells to server, Hey Server I am sending JSON data

↓

- It tells to server, what format the body data is in
- Which means, Hey, Server, I am sending JSON data
- tells the format of the body being sent
- text/plain → simple text
- application/xml → XML
- application/json → JSON data

(Request) Content-Type = what I am sending

(Response) Accept = what I am Receiving

headers: {

"Accept": "application/json",

↓

It tells the format of the body expected in Response from server

### 2) What is body?

- body is the data you send to the server in an HTTP request
- Only used in POST, PUT, PATCH
- Not used in GET

Ex: body: json.stringify({ → JSON.stringify() → Converts JS Objects into JSON string  
 title: "Hello",  
 price: 29.0  
})

4) What is JSON.stringify() ?

- JSON.stringify() converts Javascript Object into JSON string

Note:

Fetch cannot send a Javascript object directly in body

5) What is the Difference between JSON string and Javascript Object?

- Javascript Object

- Type: object
- Javascript object is a data structure in javascript used to store key value pairs
- Can contain functions, arrays, numbers, string, booleans, nested objects

Ex: Const user = {

```
    name: "Thirumalesh",
    age: 20
}
```

Note:

- Does not need double quotes for keys.

JSON string

-(Javascript Object Notation) JSON is a text for sending data.

- It looks like a JS object but actually a string

- Type: string

Ex:

Const user =

```
{ "name": "Thirumalesh",
  "age": "20",
}
```

- Cannot contain functions or undefined etc.

Note:

Must use double quotes for keys and strings values

Why we use error handling in API Integration?

\* Because, API calls are NOT Guaranteed to success every time.

\* If we don't handle errors:

- API crash

- ui may stay blank

- user won't know what went wrong.

\* API can fail - for main reasons:

- \* Network Issues:

Not internet, slow connection, slow N/W, server down

- \* Server Errors / Server Side Error

- database down

- \* Client-Side Errors:

- invalid Request Data

\* By using error handling, to prevents the api crashes and improves user experience.

\* try-catch + async/await (Recommended).

## a) Axios

What is Axios?

\* Axios is a Javascript library used to make HTTP Requests (GET, POST, PUT,

DELETE) from the server.

\* Used to connect Backend and frontend

(or)

\* Used to connect frontend with backend APIs.

\* Used to API integration

Install Axios:

npm install axios

Syntax:

axios.get(url), axios.post(url, data), axios.put(url, data), axios.delete(url)

Syntax:

axios.get(url)

axios.post(url, options)

axios.put(url, options)

axios.patch(url, options)

axios.delete(url, options)

Ex: `axios.post("http://api.example.com/users",`

`{`

`title: "iphone",`

`price: 200,`

`});`

`}); axios.delete("/products/1");`

Axios Response Object?

{

`data, → response body (Actual data)`

`status, → http status code`

`statusText,`

`headers,`

`config`

}

Note: Access Actual data:  
`res.data`

How we can create Axios instance? → custom Axios client with predefined configuration (baseURL, headers, interceptors, params etc)

```

const api = axios.create({
  baseURL: "https://fakestoreapi.com", // used to predefined configuration
  headers: {
    "Content-Type": "application/json", // Global Configuration
    "Authorization": "Bearer TOKEN"
  }
});
```

Params: {  
 category: "electronics",  
 page: 2,  
}

#### Axios Features:

##### 1) Automatic JSON handling

Convert JS Object → JSON (Request)

Convert JSON → JS Object (Response)

##### 2) Data Transformation

Transform ~~data~~, the Request and Response data automatically.

##### 3) Promise - Base:

Full support for promise API for asynchronous code

##### 4) Automatic error handling (4xx/5xx)

##### 5) Axio instance (axios.create)

Create reusable API configuration

##### 6) Interceptors

Modify Requests/Response Globally

7) Less boilerplate code / Cleaner syntax & maintenance easy

FAQs

(JSON.stringify)

In Axios, do we need body, res.json(), headers?

Ans body → X (NO)      } because axios automatically serializes request  
 res.json() → X (NO)      } data and parse JSON responses.

headers → sometimes → need only for authentication or special cases.

fetch → body + res.json() + headers ✓

axios → data + res.data + headers (optional).

Difference between fetch and axios?

fetch

Axios

1) provided by Browser (Builtin) (no install) vs Javascript library (need to install)

2) JSON.stringify() required for JSON      2) Automatic  
 request (manually)

3) ~~res.json()~~ required for JSON Response      3) Automatic (res.data)

4) NO Global Config

4) Global Config (axios.create)

5) more boilerplate code

5) Less boilerplate code

6) NO Inceptors

6) Inceptors

7) HTTP error handling (manually (res.ok))      7) Automatic

8) No Request Cancel

8) Request Cancel

9) No timeOut

9) timeOut

If the server does not respond  
 within this time, stop waiting  
 and thrown an error

Ex

```

async function postData() {
  try {
    const res = await axios.post("https://fakestoreapi.com/products",
      {
        title: "test product description",
        category: "men's clothing",
        price: 144.9
      });
    console.log("product created successfully:", res.data);
  } catch (error) {
    if (error.response) {
      if (error.response.status === 404)
        console.log("Server error - please try again");
      else if (error.response.status === 401)
        console.log("Unauthorized");
      else if (error.response.status === 400)
        console.log("Bad Request");
      else if (error.response.status === 500)
        console.log("Internal Server Error");
    } else if (error.request) {
      console.log("Network Error");
    } else {
      console.log("Error", error.message);
    }
  }
}

```

Note: Every Asyn Function always returns promise, even if you don't write a promise.

\* Both fetch and axios use GET method by default (if you don't specify the method it will automatically fetch the data or get the data).

What is HOC (High Order Component) ?

- \* HOC stands for High Order Component
  - \* It is a component that takes another component as input and returns a new component with extra features added to original component
  - \* It is a function that takes a component as input and return a new enhanced component
  - \* Component → adds extra behaviour → Return new component
  - \* Reuse common logic
- With PropTypes: → Show warnings
- \* PropTypes is a React feature used to validate the props passed to a component.
- \* PropTypes are used the expected type of props in React components and warn developers if incorrect props are passed.
- \* Prevent runtime bugs, improve code readability.
- \* npm install prop-types
- Ex: import PropTypes from "prop-types";

```
function student(Props) {
```

```
    return (
        <h2> PropTypes </h2>
        <p> {Props.name} </p>
        <p> {Props.age} </p>
    );
}
```

```
student.propTypes = {
    name: PropTypes.string.isRequired,
    age: PropTypes.number
};
```

```
function App()
```

```
{
    <>
    <student name="thiru" age=21 />
    </>
};
```

### Common Prop Types:

- 1) string (Props must be string)
- 2) number (number)
- 3) bool (boolean)

- 4) array (prop is an array)
- 5) object (prop is object)
- 6) func (prop is function)
- 7)isRequired (makes a prop mandatory)

### What is default Prop?

\* Default Props are used assign default value to props when no value

\* is passed from the parent component.

\* Prevents undefined errors

Syntax:

```
Student.defaultProps = {
  name: "thiru"
}
```

Note:- Parent can override : if parent passes props, default values are ignored.

What is the difference b/w default Props and Normal props.

(Modern Way):

```
functional Student({name="thiru",
  ...
  return()
})
```

### Default Props:

- \* Fall back values for props.
- \* used when parent does not pass a prop
- \* Prevents undefined values

What is the difference between props and state?

### Props

1) Pass the data from parent to child

2) Read-only (immutable) (Cannot be modified by the child).

3) Controlled by Parent Component

4) When Parent Props are updated then Re-render will happen

1) Internal data of Component

2) mutable (can be changed)

3) Controlled by same Component

4) When state changes then re-render will happen

## Form handling

Ways to Handle Forms in React.

- 1) Controlled Form
- 2) Uncontrolled Form
- 3) Controlled Form

- \* Form Data is controlled by React State.
- \* Input values from useState
- \* On every change → state updated

- 2) Uncontrolled Form: DOM controls the form
- \* Form Data is handled by the DOM itself.
- \* No state for each input
- \* Used useRef
- \* Used ref to access value
- \* React doesn't re-render on typing.

### FAQ

How to pass data child to parent in react?

- \* parent creates a function
- \* Parent passes that function to child as a prop
- \* Child calls the function and sends data

Ex: function Parent() {

```
const getDataFromChild = (data) =>
```

```
  {
    console.log("Data from Child:", data);
```

```
return (
  <div>
    <Child senddata={getDataFromchild} />
    </div>
  )
};

function Child({sendData}) {
  return (
    <div>
      <button onclick={() => sendData("Hello parent")}>Send Data</button>
    </div>
  );
}
```

## Component Lifecycle :

- \* Component Lifecycle means . the different stages in a React component goes through
  - \* From Creation → update → Removal
  - \* In React:-  
Mounting → updating → unmounting
- 1) Mounting phase:  
(Component is created)
- \* ComponentDidMount()
  - Runs after UI appears on screen
  - \* used for API calls, timers, Event listeners
- 2) updating phase  
(When data changes)
- \* Happens when, state; props change
  - \* ComponentDidUpdate()
  - \* Runs after component update
- 3) Unmounting phase  
(Component is removed)
- \* When component disappears from screen
  - \* ComponentWillUnmount()
  - \* used for clear timers, remove event listeners, cleanup

ComponentDidMount() → useEffect([ ], [ ])

ComponentDidUpdate() → useEffect([ ], [dependency array]);

ComponentWillUnmount() → cleanup function

## Read REDUX

### 1) What is Redux?

- \* Redux is a state management library
- \* Redux is a JavaScript library to manage app state (or) global state in apps.
- \* Keeps all data (state) in one place called a store

### 2) Why do we use Redux?

- \* Multiple components need the same data
- \* All components can access store directly
- \* Avoid prop drilling (passing the props through many levels)
- \* handles the complex shared state

### 3) Why do we use Redux instead of useContext?

useContext is good for small apps.

- \* Redux is for large apps, useContext is good for small apps.
- \* Props drilling:
  - useContext: If your app is big, you still sometimes pass props through many levels - messy
  - Redux: All state is one central store - any component can access it directly

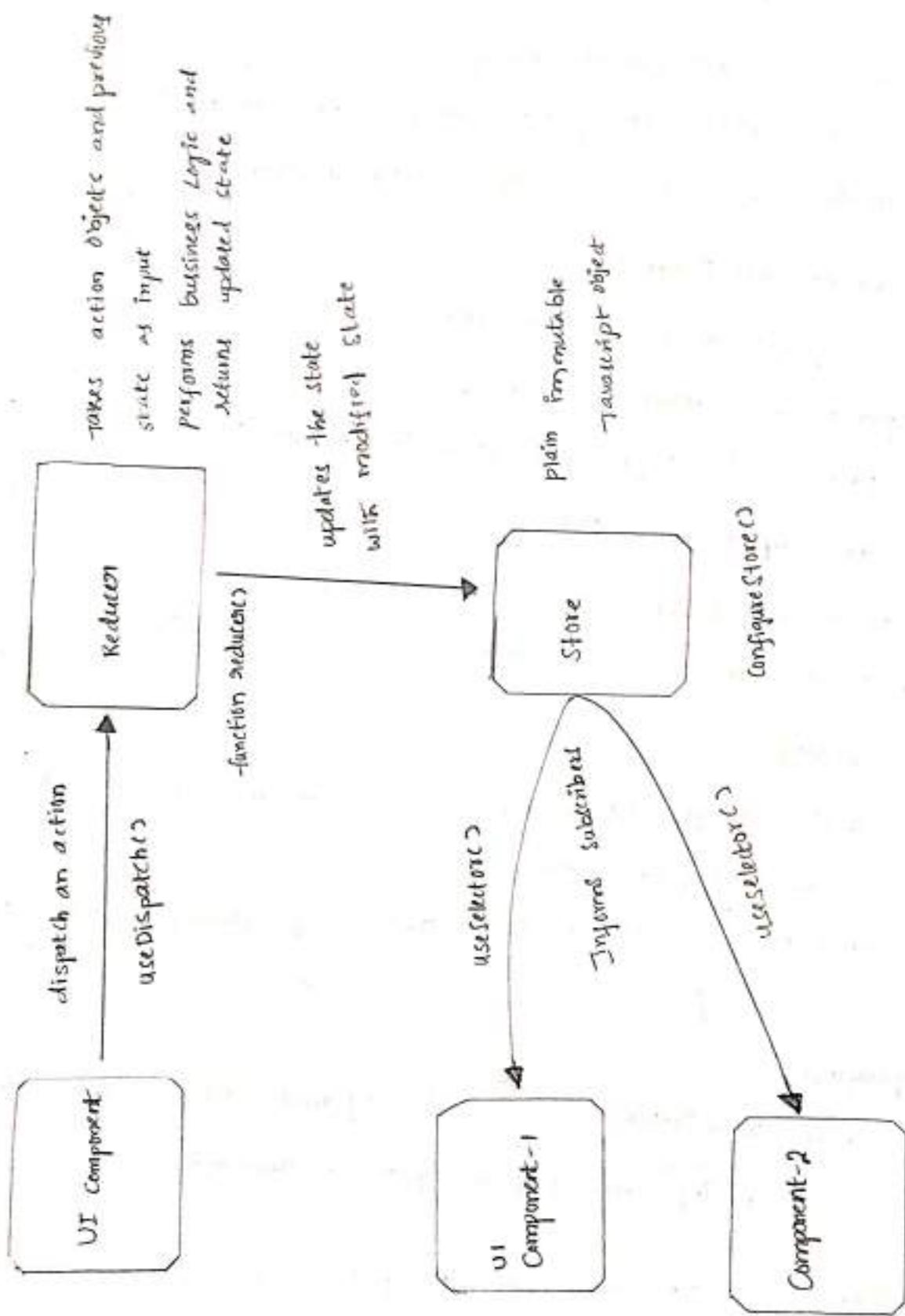
### \* Performance:

- , useContext: When context updates - all components using it re-renders, even if they don't need the update → slows app

- , Redux: Only components using that part of state - render - faster for large apps.

## Redux Terminology (State, reducer, actions, dispatch, listeners/ subscribers)

### Core Redux Terminology



## 1) Store:

- \* single source of truth
- \* holds or store data for entire application state
- \* plain immutable Javascript object
- \* Only one store in a Redux app

## 2) Action:

- \* plain Javascript object
- \* describes what happened
- \* must have a type property

Ex: `{  
 type: "INCREMENT"  
}`

## 3) Dispatch():

- \* dispatch() is a function
- \* used to send action to Redux
- \* `dispatch({ type: "name", payload: "Shivamlesh" })`

## 4) Reducers

- \* Reducer is a pure function
- \* Takes:
  - previous State
  - Action
- \* Returns new update state

Ex: `function reducer(state = {count: 0}, action) {  
 switch(action.type) {  
 case "INCREMENT":  
 return { count: state.count + 1 }  
 }  
}`

## 5) Subscribers / listeners

- \* Components subscribes to the store.
- + Re-renders whenever data changes.
- + Way for Redux to inform UI components when the store state changes.
- \* `useSelector(state => state.count);`

### Flow of Core Redux

User click a button in UI



Dispatch Action  
(UI sends action using `dispatch()`)



Reducer logic runs



store updates  
(Store Replaces Old State with new State)



UI Re-Renders  
(Store notifies all subscribed components)

(or)

UI Components

↓ `dispatch(action)`

Reducer



store (updated state).



Subscribed UI Components

## React - Redux Hooks

### 1) useDispatch()

- \* useDispatch() is a react - redux hook
- \* useDispatch() hook is used to send (dispatch) from React component to Redux store
- \* used to sending actions
- \* useDispatch() returns a dispatch function.

Syntax:

```
const dispatch = useDispatch();
```

### 2) useSelector()

- \* useSelector() is a React - Redux hook used to read data from the Redux store and subscribe to state changes
- \* Re-renders component what that value changes
- \* useSelector() returns selected state data

Syntax:

```
const state = useSelector((state) => state)
```

Ex:

```
<h1> Name: {state.name} </h1>
```

Note:

- configureStore()
- \* It is a function provided by Redux toolkit
  - \* used to create Redux store (Replacements of createStore())
  - \* combined reducers

## Core - Redux methods (Not Recommended) ✓

What is Core - Redux?

- \* Redux (core) is a standalone Javascript library for global state
- \* It works without React also

Methods in Core Redux:

- 1) createStore(): Creates Redux store
- 2) getState(): Reads current state
- 3) dispatch(): Sends action to reducer
- 4) subscribe(): Listens to state changes
- 5) combineReducers(): Combines multiple Reducers
- 6) applyMiddleware(): Adds Middleware (thunk, logger etc).

| Core Redux        | React - Redux          |
|-------------------|------------------------|
| store.getState()  | useSelector()          |
| store.dispatch()  | useDispatch()          |
| store.subscribe() | Automatic via provider |

What is Combine Reducer?

- \* Combine Reducer is a function in Redux used to combine multiple reducers into one single root reducer.
- \* Each reducer handles its own part of state.

Syntax:

```
const rootReducer = combineReducers({
  name: nameReducer,
  count: countReducer;
});
```

(01)

```
const store = configureStore({
  reducer: {
    name: nameReducer,
    cart: cartReducer
  }
});
```

Note:

ConfigureStore internally uses combineReducers when we pass multiple Reducers.

What is ConfigureStore?

- \* ConfigureStore is a function provided by Redux Toolkit to create the Redux store.
- \* Reducer boilerplate code.
- \* Combines Reducers (combineReducers) internally.
- \* Automatically adds middleware (redux-thunk).
- \* Enables Redux DevTools by Default
- \* ConfigureStore Replaces the createStore

What is middleware?

- \* Middleware is a function that sits between dispatching an action and reducer receiving it.

\* Middleware uses:

- Handle Asynchronous Code  
(API calls, setTimeout, async/await)
- perform side effects  
(call APIs)

Why do we use middleware?

- \* Reducers must pure functions

NO API calls

NO asyn code

NO - side Effects

}

- \* Middleware handles all above side Effect

Can Reducers handle asyn code?

No, Reducers must be pure functions, so middleware is required.

-----

React Portals:

What is a Portal in React?

A portal lets you render a React component outside its parent DOM hierarchy, while still keeping it in the same React tree.

Portals are useful for to avoid CSS and layout issues

Overflow: hidden, z-index, position: relative, etc.

Props and states still work.

Props and states still work.

Syntax:

```
Ex: import ReactDOM from "react-dom";
    function Modal() {
      return (
        ReactDOM.createPortal(
          <div className="modal">
            Hello modal </div>
          document.getElementById("portal-root")
        );
    }
}
```

ReactDOM.createPortal(  
)