FRONTEND

HTML
[Hyper Text Markup Language]

What is a Web?

- Internet is a wide area network that connects 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 is stands for Hyper Text Markup Language
  HyperText → It can you take the Location beyond you see.
- HyperText Refers to the text that takes user beyond the context what he view on screen.
- Markup is the term derived for "Marking", 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 (Gererric Markup Language) for internet
- CERN improved GML with force more features and introduced as SGML.
  [Standard Gereric Markup Language]
- In early 1990 Tim Berners Lee introduced HTML
- IETF [Internet Engineering Task force] developed HTML upto versions 3.1
- In early 2004 WHATWG took Responsibility of HTML and started envolving HTML
  (Web Hyper Text Application Technology Work Group)
- WHATWG started with HTML 4 (2005) 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

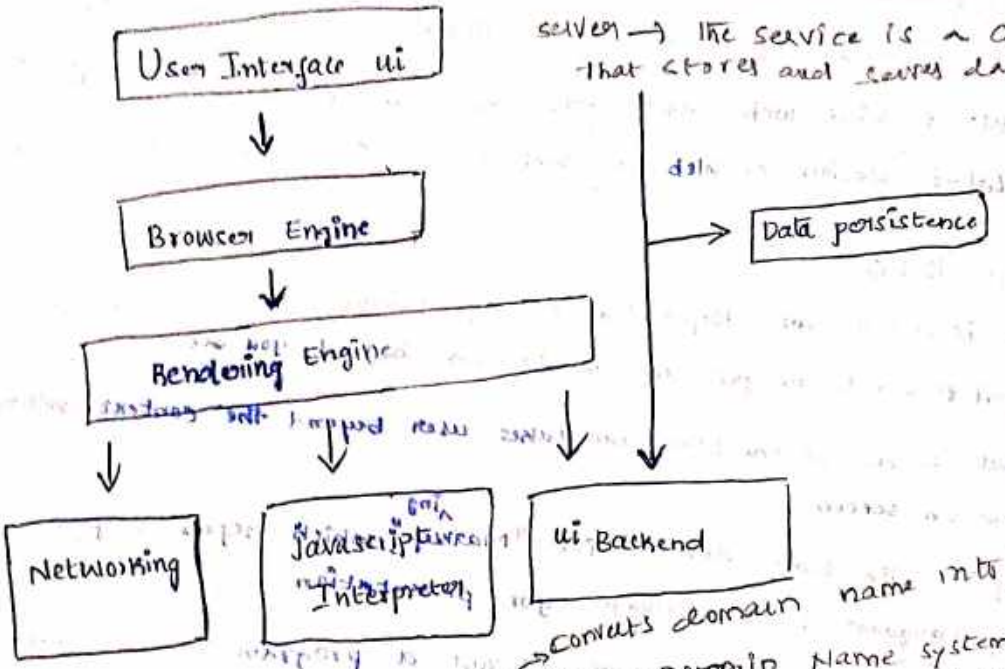
Ex: Chrome, Firefox, safari, Edge etc.

Internet → Internet is a global network that connects millions of computers which allow (text, images, video) to be shared b/w devices.

Client → client means browser (chrome etc)
- It Requests data (or) services from the server.

server → the service is a computer that stores and serves data.

Browser Architechure



Converts domain name into Ip address

DNS — Domain Name system

↳ Human - Readable address of website

(157.240.22.35)

www.facebook.com ✓

1) User Interface (UI) :

It provides the Ui for browser, which includes titlebar, menubar, short cut button, address bar, extensions, favourites 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 Intrepreter :

It is Responsible for Translating javascript line by line

6) Network :

It is Responsible for tracking the peformance of page

7) Data Persistance :

It stores data of various websites you access in browser, it includes cookies, Local storage, Session etc

Browser Engines:

- V8
- webkit
- Geko
- spider monkey
- chakra
- Nitro etc

Chrome
- Blink
- V8

Opera
- Blink
- V8

IE/E)
- Trident
- chakra

Safari
- webkit
- Nitro

Fire Fox
- Gecko
- Spidermonkey

Edge
- Edge HTML
- chakra

Website :

It is a virtual directory on web sever that provides access to various resource

Every website comprises of 2 paths

a) physical path [D : | Flipcart]

b) Virtual path http://127.00.1.5500 (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 resources of website

Tokenisation

The process of converting chars tokens into tags

- HTML is used for 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 => Bytes => Chars => Tokens => Elements => DOM => Layout => Rendor => paint

Note:
In browser developers tools you can view ~~the~~ performance

a) performance
b) Network

- HTML is a set of elements arranged in hierachy called "DOM".
- HTML elements are classified into 5 Groups.

1) Normal Elements
2) Void Elements
3) Rc Data Elements
4) Raw Text Elements
5) Foreign Elements

1) Normal Elements

- A Normal Elements 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 token.

Ex: <img>

3) Rc Data Elements

- Rc Data Elements → Rich Content Data Elements.
- These elements will not allow any other element with in the Context.
- 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 fore these Elements [Tags are not Required]

Ex: ₹

©

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 Request
- 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.
- 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

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

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

2) Every static page starts with Document Defination

<!DOCTYPE html /> → It 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-us".

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
• It is used for bookmarking page
• It used for search Engine Optimisation (SEO).

b) Link

c) Meta

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**

1) Folder "public/images"

2) Add new file into images folder by name "favicon.ico".

3) Open MSpaint in Windows

4) Open your favicon in mspaint

file = Open ⇒ d: | flipkart | public | images | favicon .ico

5) Goto file menu ⇒ image properties ⇒ set size 32×32 pixels

6) draw your icon and save [Ctrl + 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) Response
b) SEO Friendly

Meta Responsive Attributes:

1. 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 : Refers to zoom % of page [1 = 100]

2. http-equiv

- It defines how a page need to be Requested over http [HyperText Transfer Protocol]
- "http-equiv = refresh" is used to Reload a page at Regular time intervals

Syntax:

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

5 = 5 sec

NOTE: If only specific area in page is Reloading, then it 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 data bases. [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 Languages 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] UTF

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 embedded style attributes in a Web page

Syntax:

<style type = "text/css">

</style>

5) Script

It is used to embeded 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
  (b) Color shade Name
  (c) Hexa Decimal code

} In HTML colors can be defined in 3 ways.

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

(c) Hexa Decimal code :

1. 3 chars code with #
2. 6 chars code with #

#RGB [Red, Green, Blue]

#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
#fff White
#ffff White

syntax :

```
<body bgcolor = "#f00" text = "#fff">
```

for Background For text

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

Body section Attributes

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

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 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) alink : It sets color for active links in page

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

Syntax:

```
<body alink = "red" vlink = "gray">
    <ahref = "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

~~*~~ Size of web page is 1200px

Note:
Every webpage max-width is 1200 px
min-width for content must be 500 px

Syntax:

```
<body leftMargin="350" rightMargin="350" topMargin="50" bottomMargin="50">
</body>
```

BODY SEMANTICS:

* Semantic Refers to element ment for a specific purpose.
* The name defines what is 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. <nav>
6. <article>
7. <aside>
8. <aside>
9. <dialog>
10. <figure>
11. <figcaption>
12.
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 topmargin of page
2. <footer>	: Content to display at bottom margin of page
3. <section>	: It defines the content display btw 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 configure a pop-up.
9. <figure>	: It encapasulates any image (or) graphic with caption
10. <figcaption>	: It sets SEO friendly caption for image.
11.	: It is a division
12.	: It spans the content along with existing content, however it can seperate 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>
```

3) Apply icon using or with bootstrap icon class name

1) Header:

Header comprises of brand name, brand logo, shortcut buttons, query, search etc.

Css Text Styles:

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

Css Display Flex:

display : It keeps the element flexible in container
your 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.

2) Nav

- It is used for configuring navigation area in image
- Items in nav can be defined using

Syntax:

```
<nav>
<span> Home </span>
<span> About </span>
<span> ContactUs </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>
```

3) aside :

- It defines the content which is not Relative to current context.
- It navigate the user to partner bussiness

Ex: Advertisements.

Syntax:

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

Css Abosolute and Fixed positions :

1) Position : absolute → It Removes element from its position and keeps with Relative to content in page. It will scroll along with Content.

2) position : fixed → It Removes elements from its position and keeps Relative to brawser. It is locked with scroll.

3) top, left, right, bottom → These are the used for define the position of the element in page

Syntax:

```
aside
{
position : absolute | fixed;
bottom : 10px;
right : 1px;
}
```

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

If you dont want scroll (or) Locked the Icons then you will use it

4) Section :

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

5) 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 => 0 to 255

alpha => 0 to 1

Ex: rgba (0, 255, 0, 0.6) => Green transparent color

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

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

vh => view port height 100 = 100%.

vw => view port width 100 = 100%.

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

8) Dialog

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

Syntax

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

9) Figure

- It is use to display the images and Graphics with caption
- "Figcaption" 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 developers tradition to define the Self ending elements [void Elements].

Functionality
 and
 will be same.

2) Blank spaces

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

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

syntax:

Words   Words.

3) Performed Text:

- It keeps the formats in page exactly as defined in editor.
- Performed 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 values (like in java), it just gives semantic meaning to text.

Syntax:

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

5) Sample Text:

- It is used 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 palindrone "</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>
. . . . . .
. . . your content . . . .
</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?

Soln 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 styles?

Soln yes. It includes font-styles, size etc.

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

Soln 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 to many headings in web page. it may SPAM (unwanted messages)

2) dont use headings for highlighting an word a sentence in a paragraph, it may cause of voilation 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?

Soln 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 styles?

Soln yes. it includes font-styles, size etc.

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

Soln 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 to many headings in web page. it may SPAM (unwanted messages)

2) dont use headings for highlighting an word a sentence in a paragraph, it may cause of voilation 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)
- Blockquotes is similar to paragraph but have left and Right padding.
- Blockquotes used to summarise the content in the page. <blockquote>.
- paragraphs and Blockquotes designate a container, which can be aligned in page.

Syntax:

```
<p align = "centre"> Some text </p>
<blockquote align = "centre">
    ....
    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 wit <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> Defination </dd>
</dl>
```

FAQ:

1) HOW to set Dropcap for paragraph ?

Sol By using css class "::first-Letter" and css attributes like "size and float".

Syntax:

```
p::first-letter
{
font-size : 55px;
font-family : Algerian;
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 {
  text-indent : 200px;
  }
```

9/9/2025

LISTS :

- An HTML List allows you to organise 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 usefull for creating structured, accessible content in web dev.
- tag is used to define each item of the List

Syntax:
 Content

Types of HTML Lists:

1) Ordered List (<ol>) : Items in a list that are listed in Order of numbers or Letters

2) UnOrdered List (<ul>): Bullet items which donot have a natural Ordering

3) 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 various list items in a Order.
- It sets auto numbering for List items
- the Ordering is given by a numbering scheme, Arabic numbers, Letters, Roman numerals.
- List is defined by
- Items are defined by

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="51">
```

* start is 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>
  <ol type "a>
    <li> child </li>
    <li> child </li>
  </ol>
</li>
</ol>
```

FAQ)

1) 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
}
```

2) how to keep list items in multiple columns ?

By using 2 styles attributes

a) display : grid sets numbering order left to right

b) columns : sets numbering order top to bottom

Grid :

1 2

3 4

columns :

1 3

2 4

3) how to create a scrollable list ?

By using following css attributes

- border
- height
- overflow

syntax :

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

4) how to remove 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.
- It is also called as Bulleted List.
- These are very popular in webpages where they are used in displaying menus, feauture, sets, grouped information etc
- An unOrdered list tag is used to create an unordered list
- Item are defined by <li> tag
- An unordered 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 unordered Lists:

- type = "disc" - sets the list item marked a bullet (default).
- type = "circle" - sets the list item marker to a circle.
- type = "square" - sets the list item marked to a square
- typ= "none" - the list items will be 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 20 x 20 px in ms paint
→ draw and save your icon as a Gif type
set for

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

Description List :

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

- this contain <dt> and <dd> tags.

2) <dd></dd> → It is used to describe the term
1) <dl></dl> → It is used ~~used~~ to defined the term

Text Formating in HTML :

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

- HTML provides ~~w~~ ability ~~w~~ ability to format the text without css.

- Text Formatting allows you to style text in different ways, such as the making it bold, italic, underlined, highlighted.

physical Tag : these tags are used to provide the visual appearance to text
Logical Tag : these tags are used to add some Logical or semantic value to the text

Element name — description

1) <b> → this is a physical tag, which is used to bold the text in written between it

2) → this is a logical tag, which tells the browser that the text is important

3) <i> → this is a physical tag which is used to make text as italic

4) " " " logical tag " " display content Italic.

5) <ins> → It is used displays the content which is added

6) <del> → It is used to ~~delete~~ display the deleted content

7) <sup> It display the contents slightly above the normal line.
8) <sub> It display " " " below " " ".
9) <big> " " " increases the font size.
10) <small> " " " decreases " " ".
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" noshade>
↳ specifies the bar without shading 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:

<img src="url" alt="description">

Web supports only following various types of image.

1) APNG - HD Image	image/apng	Ammated portable Network Graphics [.apng]
2) PNG	image/png	Portable Network Graphics [.png]
3) JPEG	image/jpeg	Joint photographic Expert Group [.jpeg]
4) GIF	image/gif	Graphic Interchange format [.gif]
5) TIFF	image/tiff	Tagged image file formate [.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	We b picture [.webp]

FAQ):

When to use Specific image type? what are the its feautures & applications?

APNG & PNG - HD images, more pixel depth, but it occupies more memory
- If you are defining images for download then prefer png.

JPG - HD image, Compressed images, Less memory space.
- If you want display images directly on the the page then use jpeg

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 Resultion and it is a xml based image

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

TIFF, Bmp - Not mandatory, used in documents
used for Binary formate image

Ico - It is used for icons [favicon]

Attributes:

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

Syntax:

<img src = "images/pic.jpg">

) alt : it defines the ~~text~~ alternative text to display when image fails to load.

syntax:

<img src = "imges/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 = "imgs/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 = "100" 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 = "100" height = "400" align = "centre">

6) Vspace & hspace : It sets horizontal and vertical space between text image.

syntax :

~~<imag~~ <img src = "images/ pic.jpg" align= "right" vspace="40" hspace = "40" >

Advanced Attributes:

1) CrossOrgin [CORS] - CROOS - Orgin Resource Sharing

- used when loading images from another domain (like a CDN)
- controlls whether to send credidentials (cookies, HTTP auth) when fetching the image

syntax :

<img src = "... cdn ... path ..." crossorgin = "user-credidentails | anonymous ">

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

it have the techniques : sync, asynt, auto.

syntax :

<img src = "images | pic.jp" decoding = "sync | asynt | auto">

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

syntax :

<imag src = "images | pic.jpg" importance = "low | medium | auto | high ">

4) srcset : It is used to define a set of images, which can be 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 "<a>" tag which stands for "anchor".
- Hyper Links are classified into 2 types.
  1) Intra Document Links
  2) Inter Document Links.

1) Intra Document Links. [Links within the same Html document (page)]

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

syntax:

 Visit Google

href → HyperText REFerence
↳ It is a attribute to specifices the destination.

- Configuring intra document links comprises of 2 phases

1) defines (ID) attribute for targeting element
2) Navigate the ID by using <a> href attribute

syntax:

<h2 id = "html">
<img id = "realme">
 html

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 unvicited 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;
}
```

2) 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 alink 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:

 Go to page1

FAQ

How to Configure a link that downloads the given file?

By using "download" attribute for <a> element.

Syntax:

 Fashion

How to open linked document in new window?

By using 'Javascript' window.Open() method

Syntax:

 Text

  Aphoto .

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 display the a nested webpage (a webpage within a page). the html <iframes tag defines an inline frames, hence it is called "inline frame.

Syntax:

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

- We can it used for embed image, documents, audio, video etc.
- We can configure a link that navigates to any specified application

Syntax:

 Email
" " "skype: some@Outlook.come"> Skype
" " "tel: +91 9705 70041"> Call

mail → Opens mail application
Skype → " skype app
tel → " phone "

Entities in HTML

A Text fragment (or "string") that starts with an ampersand (&) and ends with 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: Σ - ₹
  - nDn-breaking Space

&lt → <(lessthan)
&gt → >(greaterthen)
&dollar → dollar ($)
&copy → ©
&trade → trade

⇐ → ⇐
≠ → ≠

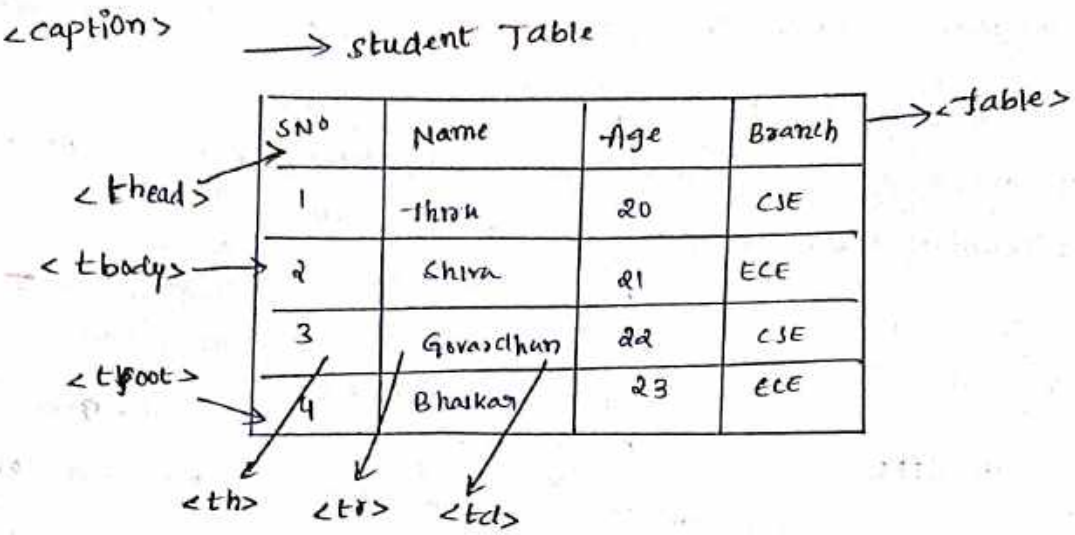
Tables in HTML

- Tables are used arrange the content in the form of rows and colourmns
- 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 simple text 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.

9) <tfooter> : It is used to group the footer Content in table



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 a 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 be 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/ resources where form data will be submitted.

6) novalidate : HTML5 have default validations, which you can disabled it. (required, type = "email", pattern etc)

Syntax :

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

Form Elements

- Form Elements are the interactive control inside <form> tag. that let users enter and sends 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
drop down etc.

Html provides different tokens for creating elements.

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

Note:
All the elements must be the inside the page -form Container. if element are configured. Outside <-form> then page cannot submit their data to server.

HTML Forms Tags:

1) <input>

. <input> tage in HTML is used to collect user input in web forms.

. It facilites 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 forms

Syntax:

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

3) <select> :

- <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 Retrive user Responces.

syntax:

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

4) <Option>

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

syntax :

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

5) <Optgroup>:

- <Optgroup> is a tag which is used to group related <option> elements within a dropdown list
- Which specifices 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
- usefull 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> tags 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)

Syntax:

```
<textarea name = "text" rows = "10" cols = "50">
Enter text here .....
</textarea>
```

Attributes:

1) name: name of the field (text,)
2) rows: Number of visible text lines
3) cols: width of the textarea (in characters 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, downing, loading etc

Syntax :

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

</progress>

Attributes:

1) max → It is Represents total work is to be done to complete task.

2) value → It Represents the amount of work that is already completed

9) <meter>

- <meter> tage 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 .... >

</meter>

Ex: <meter value="5" min="0" max="100">

5 out of 10

</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, running, Javascript functions, or simply interacting with users
- <button> tag Contain 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">

</label>

Attributes:

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

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 elements
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 = "ghiru" disabled >

7) placeholder:

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

Syntax:

<input type="text" name="username" placeholder="enter your username">

8) autofocus

It sets focuses 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="thirty" 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 chars 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>.
- works with "text, search, url, tel, email, number"

Syntax:

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

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

A Regular Expression is built by using

a) Meta Characters

b) Quantifiers.

a) Meta characters

Characters	description
?	Zero or one occurance of character Ex: colou?r => color, colour
*	Zero or more occurance of character Ex: colou*r => color
+	one or more occurances of character ex: colou+r => colour, colouur

. (dot)	any single character
	Ex: .at => cat, bat, 2at, #at
	.t => .ot, cat

|	logical OR, any 1 of gives sets of value ex: pattern : "green|blue|red"
\	It is an escape sequence character it changes a normal character not meta character vice versa.
^ - starts with	It defines excluding specified chars starts with ex: pattern = "^d" ⇒ allow only "d" and exclude others
\n	It defines starts with
$ - ends with	It defines ends with ex: pattern = "|^ expression $"
( )	It defines union of chars or expressions.
[ ]	It defines the Range of characters or multiple chars set [collection]

Ex: pattern = "[a-d]" → a,b,c,d
" = "[^a-d]" → all other than a-d
" " [a,f,w] → only specify allowed
" [^a,f,w] → all others allowed excluding "a, f, w".

\d	any single digit number [0-9] ex: pattern = "[0-9]"
\D	It indicates non-numeric (all characters other than number) ex: pattern = "\D" → [a,z, A-Z] and special characters are allowed ex: pattern = "\D\d" → A3 ex: pattern = "\d\D" → 3A
\s	It Represents single blank spaces.

Note: Never usually directly ^ blank space in pattern always configure with "\s".
ex: pattern = "\D\d\s" → A 4, $ 5

\w (small letter)	It defines a word char word char can be [A-Z, a-z, 0-9] ex: pattern = "\w" ⇒ [A-Z, a-z, 0-9] ex: pattern = "\w\d" ⇒ [a4, -4, A5] ✓ only special chars are allowed other than _ (underscore)
\W (Capital letter)	ex: pattern = "\W" ⇒ #!@%.^*&.

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

{n} exactly "n" number of chars

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

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

FAQ 17
Write the Regular Expression to validate Ifsc code SBIN0000@345

<input type="text" name="Ifsc" pattern = "SBIN0000\d{4}

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

<input type="text" name="mobile" pattern = "\+91\d{10}" >

Write a Regular Expression for to validate US validate +(1) (425) 555-0100

<input type="text" name="Ifsc" pattern = "\+(1)) \((\d{3}\)) \s\d{3}-\d{4}" >

- You can implement pre-defined Expressions.

(?=.*[A-Z]) atleast one upper case letter

(?=.*[a-z]) " " Lowercase "

(?=.*[0-9])

(?=.*[!@#$%^&])

- We can use css validation classes

1) :valid → applied when the input values matches the pattern

2) :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" >
```

3) 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(protocal) :// 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

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

<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 use Mutex Manner

Syntax:

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

- you can make the radio selected by default by using "checked" attributed

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

8) Date & Time

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

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

* you can configured the data picker with Restricted data and time Range, by using below attributes.

- min
- max

* the data & time for time format developer is :

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

< syntax :

```
<input type = "date" >
<input type = "time" >
<input type = "date time-local" >
<input type = "month" >
<input type = "week" >
<input type = "year" >
```

Syntax: for Restricted [the date and time which means should select particular [illegible]

<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 is a displays a color picker, that allows user to pick a color.

* you can configure default color by using only 6 char hexa 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 = 37 ✓)

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.geeksforgeeks"
height = "50" width = "30" >

13) reset

· It creates a button that reset all from fields to their default values

· It only resets to original state

Syntax:

<input type = "reset" >

14) Search

It is used to search boxes, this types encourages users to type and initiate search operations

Syntax:

<input type = "search"

15) tel

It is designed for telphone 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

It 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 to you can user for selection one or multiple

Syntax:

<input type = "checkbox" name = "Courle" value = "java" checked > java

note: checkbox submits "on" as value if "value" attributed is not defined

FAQ): How to design a checkbox list?

HTML dont 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

It is a new class in css used to verify the checked status of any checkbox or radio.

Syntax:

```
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"> ⟹ value is Rc data type

<input type = "reset" >

<input type = "sumit" > text | image |

~~<input type~~

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

note: if button type it not defined html button, then the default type is submit

<button > click </button> ⟹ 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 | Graphics </button>

Form Controls

1) DropDown

- Dropdown, Commonly known as <select> elements,

It allow user to choose one or more options from a predefined list

Syntax:

```
<select>
  <option value = "Option 1"> Option1 </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

3) 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 RCdata type, you cannot present images or symbols.
- It is only for plain text and special characters and it will not allow markup inside option.

2) ListBox

- It allow user one or more multiple options from a group of choices.
- a dropdown is tranformed 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
- textbox allows only single line input, but textarea is requires to accept input in multiple lines.

Syntax:

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

rows → visible no. of lines (height) ; cols → width of the terms of any character count

- placeholder → show a text inside the box until the user types
- readonly → user 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 gracle meter in webpage

Attibures:

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

Syntax:

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

**Note

if low and high are 0 then Color → Green

if low and high difference is less then Color → Red

if low and high difference is more then Color → yellow ✓

5) Progress

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

Syntax:

<progress id="arial-multiline" max="100" value="100"></progress>

() Label

- It is 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 = "user name" > 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) Marque

It is used to set scrolling content in page.

Syntax:

```
<marquee>
... your Content ...
</marque>
```

Attributes:

1) scroll amount : speed of scrolling (1 to 100)
2) direction : Controls scrolls direction (Right, left, up, down)
3) behaviour : Changes scrolling to sliding or type of movement
   - scroll → default
   - slide → 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 marque area.

6) bgcolor : sets background color.

Syntax:

<marque direction = "left" behaviour = "alternate" scrollamount = "5" loop = "3"

</marque>

Note:

You can pause and Resume Marquee by using javascript functions

<marquee onmouseover = "this.stop()" onmouseout = "this.start()">

2) Video

- <video> Tage 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 automatially (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" control> </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

CSS (Styles)

16-9-2025

* 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 users how to display Html elements on the screen in proper format.

Applications of CSS:

1) Styling Web pages:
2) Responsive design → make website adapt to different screen sizes [desktop, mobile]
3) Animations and Transitions
4) Website management
5) Accessibility Improvement
6) Ecommerce
7) Social Media.

Css Versions:

Css1 → 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, Boards, 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 </h>.

2) 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 : centre;
    }
 </style>
 </head>
 <body>
  <h1> Internal CSS Example > </h1>
  </body>
```

3) External CSS: (Recommended)

Styles are written in a seperate .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 : centre;
    }
```

FAQ)

1) How to add css file to Html by External css?

Ans 1) Create a file with .css extension eg; style.css

2) Write your css Rules inside it

3) Links the css file in your html

→ In head section, use <link> tag

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

2) 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) Rational selectors
3) dynamic selectors
4) Universal selectors
5) Element state pseudo
6) validation states pseudo
7) Structural pseudo
8) Root, Languages selectors.

1) 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 single a select, unique element
- Every element can have only one ID Reference

Syntax:

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

c) Class Selector:

- A class selector is defined by using ".refName" (or) ".classname"
- class selector is defined with "." (dot)
- It is accessed and Rended using "class" attribute.
- Every element can implement multiple classes.

2) Universal selector

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

Syntax:

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

4) Group selector:

- the Group selector is used to select all the elements with same style definations
- Group selector is used to minimised code. , (Commas) are used to seperate each selector in Grouping

Syntax:

```
h1, h2, P, body {
        text-align : centre;
        color : green;
        }
```

5) Rational selectors (Combinators)

these define Relationships b/w elements.

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

Syntax:

```
ul li {
}

nav div {
}
```

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

Syntax:

```
nav > div {
}
```

3) General sibling: It can aucss all elements that are designed after specific it is configured with "~"

Syntax:

```
A~b {
}
```

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

Syntax:

```
Ae+Be {
}
```

5) structural pseudo class:

used for selecting elements based on their position in DOM.

: first-child → fild child element
: last-child → last child element

:nth-child() – specific child element you can define even or add.

: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}

:nth-last-of-type(1) → (last li) ✓

{color: orange}

7) Attribute selectors

[attr] → elements with attribute

[attr="value"] → exact match

[attr^="value"] → starts with

[attr$="value"] → ends with

[attr*="value"] → Contains anywhere

[attr|="value"] → stats 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 Lastter
::placeholder → style the placeholder text
::firstline → sty the first line
::selection → style highlighted text

:: → Inheritance

9) Element state Pseudo classes

a) :enabled → Targets form elements that are enabled (default 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 elements focus

d) : blur [obsolete - no longer in use] → effects when element lost focus.

f) readonly → when element set to readonly

g) : option → matches with inputs "required" attribute.

Syntax:

input : optional {

border : solid gray

}

h) valid : → matches the input fields whose value meets validation rules (email correct

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

. Work 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 on mouse down (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 brower.

Syntax:

/* this is CSS single line Comment */

/* this is a CSS
Multiline comment */

Styles with CSS

In Colors

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

Color formates 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) → 0 (transparent to 1 (opaque)

Syntax: rgba → red, green, blue, alpha → 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 → degree on color wheel (0-360)

  0-120 → Red shades

  120-240 → Green shades

  240-360 → Blue shades

- Saturation → intesity (0% = gray, 100% = full color)
- Lightness → brightness (0% = black, 50% = normal, 100% = white).

Syntax:

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

d) Gradiant Colors → smooth blending for multiple colors

Configured with back-ground-image

a) Linear Gradient

```
body { background-image: linear-gradient(to right, red, green 70%, 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-~~position~~ attachment → Defines the background image scrolls with page. (scroll, fixed, local)

Syntax:

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

(or)

```
body
{
background: Green url("bg.jpg") no-repeat fixed centre top;
}
```

III) Text styles

1) Color: set the text color

```
P {
Color : green;
}
```

2) text-align : Align text horizontally
values = (left / Right / centre / justify)

Syntax:

```
h1
{ text-align : centre;
}
```

3) text-decoration : Adds or Removes decorations from text

Values = none / underline / overline / linethrough

Syntax:

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

4) text-Transform : Change case of text

Values : uppercase / lowercase / capitalize.

Syntax:

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

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

Syntax:

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

5) letter-spacing : control spacing between charaters.

```
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;
     }
```

2) font-family

- Defines which font face to use.

Syntax:

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

3) font-size:

- Sets the font size

syntax:

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

4) font-style

- Defines if text should be normalic, italic, oblique etc

syntax:

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

5) font-weight

- Control how bold or light text is

syntax:

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

V) 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)

Values:

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

Syntax:

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

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 of Color

Syntax:

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

or

```
div
{
border-color: red green blue yellow;
} /* top, Right, bottom left */
```

4) border - radius:

- Rounds the corners of the border

Syntax:

```
div
{
 border - radius : 15px;
}
```

VI. 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.
 margin - top : 15px ;   "  "  "  on the top of element
 margin - left : 15px;   "  "  margin on the left side.
 margin - bottom : 15px →"  "  "  on the bottom of element
}
```

VII 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;
}
```

shortform →

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

Css units or measurments in css:

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

2) % perencentage
- 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

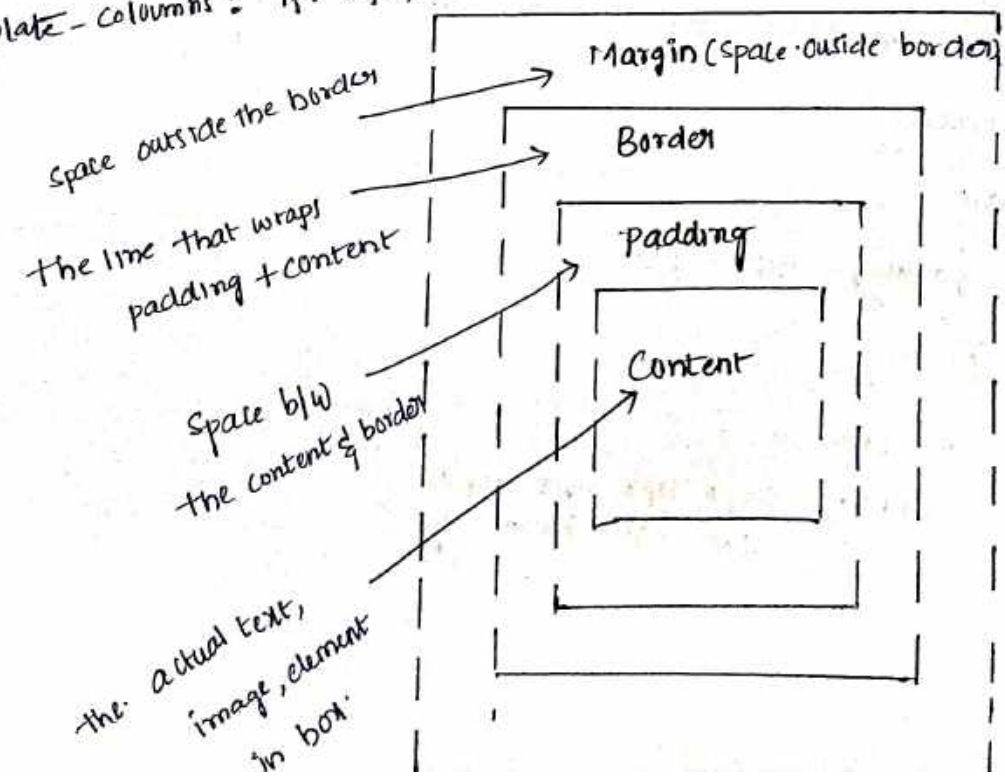
syntax:

```
.hero {
    height : 100vh;
    width: 100vw;
}
```

4) fr (fractional unit in Css Grid)
- Represent the portion of available space in grid layouts.

syntax
- container {
    display : grid;
    grid-template-columns : 1fr 2fr;
  }

Box Model:



(VIII) Display properties

* It specifies 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;
   }
```

XI. Flexbox Properties

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

syntax:

```
. container {          || Enabling
  display : flex
  }
```

1) Flex Container Properties

a) display : flex → defines a flex container

syntax:

```
. container {
    display : flex ;
    }
```

b) flex-direction :

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

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

Syntax:

```
. Containtar {
  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 / wrapreverse.

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 ~~distributed~~ inside the container

Values: flex-start / flex-end / space-between / space-even / space-around.

syntax:

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

d) Align-items

Defines the vertical Alignment of flex ~~*~~ items inside the container.

syntax:

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

2) Flex Item properties

a) Order

control order of items

```
.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

define how much items shrinks when not enough space

```
.item {
  flex-shrink : 1
}
```

d) flex-basic

define intial size of item

```
. item {
  flex-basic : 200px;
  }
```

X Grid System Properties

Grid Layout allows to divides the webpage into the Rows and colounms

Grid Container Properties.

a) display : grid

It can defines the element as the grid Container

syntax:

```
. container
{
display : grid;
}
```

b) grid-template-columns :

It can specificies the number and size of the Colounms in the grid

syntax:

```
. Container
{
display : grid;
grid-template-columns : 200px 200px 200px; / 3 Fixed Colounms./
}
```

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-grap

It can specifies the spaung b|w the Rows and Columns

Syntax:

```
.containter
{
display: grid;
grid-grap: 10px;
}
```

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

XI 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 position 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 : Stickly

- 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 scroll bar can be horizontal or vertical

Syntax:

```
overflow: scroll;
```

overflow-x → horizontal

overflow-y → vertical

Ex:

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

d) Overflow: auto

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

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

<link href = "../node_modules/bootstrap/dist/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.

a)
- container
- Fixed width container
- 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 720 px, 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)

* Fluid width up to 1140px, Fixed width above 1140 px.
* 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
d) .m-{size} → size can be 1 to 5

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

Ex:-
M-4
syntax :-
  // 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
d) .p-{size} → size can be 1 to 5
syntax:
<div class = "pt-3 > // 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 contextual} → changes the border color.

Note: primary → Blue
secondary → Gray
Success → Green
danger → Red
Warning → yellow

Syntax:

(success → Green)

e) . rounded → slightly Rounded corners.

f) . rounded-circle → perfect circle

g) . rounded-pill → Fully Rounded

h) . rounded-{size} → Increasing the Rounding size

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

syntax:

5) Width and Height classes

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

→ useful for images, buttons, forms, boxes, div etcs.

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, forms, and interactions.

a) . btn : base class button
b) . btn-{contextual} : primary, danger, warning, secondary, success.
c) . btn-link → Looks like a link
d) . btn-sm → small button
e) . btn-lg → Large button

f) • btn - close → close button

g) • btn - group-vertical → Buttons are in vertical

h) • btn - group - horizontal → Buttons are in Horizontal

i) • btn - toolbar → Group Multiple Group together in a single Horizontal line

Syntax:

7) Text and Background classes:

a) • text - {contexutual} → primary, Green, success, warning, danger

(Changes the text color)

b) . bg - {contexutual} → primary, Green, success, warning, danger.

(Changes the Background color).

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

Ex: i) Syntax :

ii)   // small text in centre.

d) • text - uppercase → text with upper case

e) • text - lowercase → text with Lowercase

f) • text - Capitalies → text with capital

ε) rows and Columns

* • row is used to create a horizontal group of columns

* It ensures proper alignment & spacing.

Syntax:

* • col is used to create Responsive, equal width columns.

* Bootstrap uses 12 - column grid system

Syntax:

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

9) Display Content centred on screen:

* to Center Content both horizontally and vertically, uses Flexbox utilities in Bootstrap

a) . d-flex → makes the Container a flex Container
b) . justfy-content-centre → Centrens content horizontally
c) . align-items-centre → 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 → standards input for file
e) . form-select → dropdown/select menu
d) . form-checkbox-input → For checkbox or radio button
e) . form-check-label → label for Checkbox / Radio
f) . 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 Effating its functionality.

* It Improves performance and Reduce load time.

Ex:- /* Original css */

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

/* Minified css */

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

2) What is CDN ?

CDN is (Content Delivery Network) a Distributed Network of servers that hosts libraries like (CSS, js).

Css Rules and priority:

FAQ)

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

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

2) 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

24/9/25.

* 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 messege box used to display notifications, warnings, success message or errors.

Classes:

1) .alert → Base class for creating an alert.

2) .alert-{contextual} → colors.

Ex:- .alert-primary (info messege) (Blue)
.alert-success (success message) (Green)
.alert-danger (error/danger messege) (Red)
.alert-warning (Warning messege) (yellow).

Syntax:

 this is a succes alert msg .

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

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

Syntax:

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

2) Modals:

* Modal is a dialog box that pops up in page

(or)

* Modal is a dialog box/ popup that appears on top of the current page content

* Hidden by default, shown using Javascript/ jQuery attributes.

classes:

a) .modal → Base class

Syntax :

b) .modal-dialog

Defines the Modal box position.

c) .modal-dialog-centred

Vertically centres the modal

d) .modal-dialogue-scrollable

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

e) .modal-dialogue-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 button or avatar or image etc

h) .modal-body :

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

i) modal-footer

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

ii) • 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 = "model-footer">
        </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 Badges, A small lightweight component used to display counts, labels, highlights.

• Commonly placed next to buttons, headings, links.

Classes

a) • badge → * The small count or notification dot.
* Base class of all badges.

b) • badge-{contextual} → Background colour
CEx: bg-warning , bg-primary etc).

c) .text - {contextual} → Text color

ex :- Text-primary, text-success etc

Syntax:

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

4) Cards

* A card is a flexible and extensible content 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
Subtitles 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
place text on top of an image

5) Spinners

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

classes:

a) . spinner-border
* creates a border-style rotating spinner (Defaut)
* Defaut it is hallow 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
Larger version of grow spinner.

h) . text-{contextual}
Changes the spinner color (primary, secondary, danger, warning, success, etc)

syntax:

 // For Borders spinner

 // 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 completion

Classes

a) .progress

* Base class of all progresses.

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 continously, giving the effect of an active process.

* usually combined with .progress-bar-stripped.

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-stripped prograss-bar-Animated"
     style="width: 30%"> 30% completed.
</div>
</div>
```

7) Carousel → It slides automatically or by clicking arrows/dots

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

Classes:

a) .Carousel → Main Container / Base class of all carousels.

b) .carousel-inner

* A container inside .Carousel

* Holds all the slides (items)

c) .carousel-item

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

d) .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 not start with any item by default. you have to set any one carousel -item is "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-ride";

Syntax:

```
<div class="carousel" data-bs-ride="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 carousel-fade">
```

Carousel controlls:

* they allow to navigate b/w slides
* you can move to previous or next

classes:

a) .carousel-control-prev → previous button (←)

b) .carousel-control-next → next button (→)

c) .carousel-control-prev-icon :

* the icon (arrow) inside the previous button.
* Bootstrap gives a default left arrow.

d) .carousel-control-next-icon :

* the icon (arrow) 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 :
1) "prev" → move to previous slide
2) "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-indicators → 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 timings 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:

 .

3000 millisec = 3sec.

8) NavBar

* A Navigation bar is used to create menus (Home, About, Contact etc).
* It is Responsive → automatically collapses into a hamburger menu on small screens.

Classes:

a) .navbar
* Main Container for the NavBar

b) .navbar-expand-lg | md | sm
* Controlls when navbar collapses.
Ex: .navbar-expand-lg → expands on large screens and up
.navbar-expand-sm → expands on small screens 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.

j) bg-* or bg-{contextual}

* → primary, secondary etc.

* sets background colors for Navbar

9) 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

10) 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 → Optional footer for each item

attribute:

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

11) 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 dropdowns items

g) .dropdown-item → Each clickable option inside the menu

h) .dropdown-divider → Adds the horizontal seperator b/w items

i) .dropdown-menu-dark → Dark-themed dropdown menu

JAVASCRIPT

27/9/25
saturday

* JAVASCRIPT (JS) is a programming language thats developers create interactive web pages.
* Javascript is a lightweight, Object Oriented programming Language.
* Javascript is a JIT-Compiled and Intrepreted programming Language

Intrepreter → 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 ~~ESTAA Script 2025~~ 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.

Feautures of Javascript

1) Lightweight
2) Dynamic Typic
3) Object-Oriented Programming (OOP)
4) Procedure-Oriented
5) Intrepreted 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 " from "
- Update Content in elements
- Dynamically configure styles

b) Browser Interactions
- Detect browser versions & Compatibility
- manage plugins & Extensions
- Access Client location & History
- perform browser window operations

c) validations
- Form validations before sending data to server
- prevent unneccessary server requests.

Evolution of Javascript:

1) ECMA script 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 Microsystem, which Renamed it "Javascript".

4) 2002 : Net-scap stopped developed and transformed javascript standardrazation 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 seperate .js file
- allows Reausability 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 ?

Sol 1) By using DOM Hierachy (Index Numbers)

2) By using Reference Name

3) By using Unique ID

4) By using css Selectors

1) DOM Hierachy → HTML as Objects + Tree structure + javascript can control it

* DOM → Domain Object Model → It is an interface
* DOM hierachy is a tree-like structure that Represents an HTML or XML document.
* DOM is the Fastest 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:

1) 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";
```

2) getElementByTagName( )

Returns list of all Elements with the specified tag.

Index-Based (Returns HTML collection).

selects all tags of same type.

→ Affects when by the position change

Syntax:

```
document.getElementByTagName("tagName")[indexNumber];
```

Ex:

```
document.getElementByTagName("img")[0].src = "photo.jpg";
```

3) getElementByClassName( )

* Returns list of all elements belongs to the specified class.
* Index-Based Required
* selects all elements with same class.

syntax:

```
document.getElementByClassName("className")[indexNumber];
```

Ex:-

```
document.getElementByClassName("box")[1].style.background = "Green";
```

4) getElementsByName( )

* Mostly used in forms
* Returns NodeList → index Based

syntax

```
document.getElementByName('name')[index];
```

Ex:

```
document.getElementByName("username")[0].value = "shiva";
```

5) QuerySelector() (Recommended)

* Selecting the First matching element
* Returns Only One element.
* Returns the First object matching css style selector

Syntax:

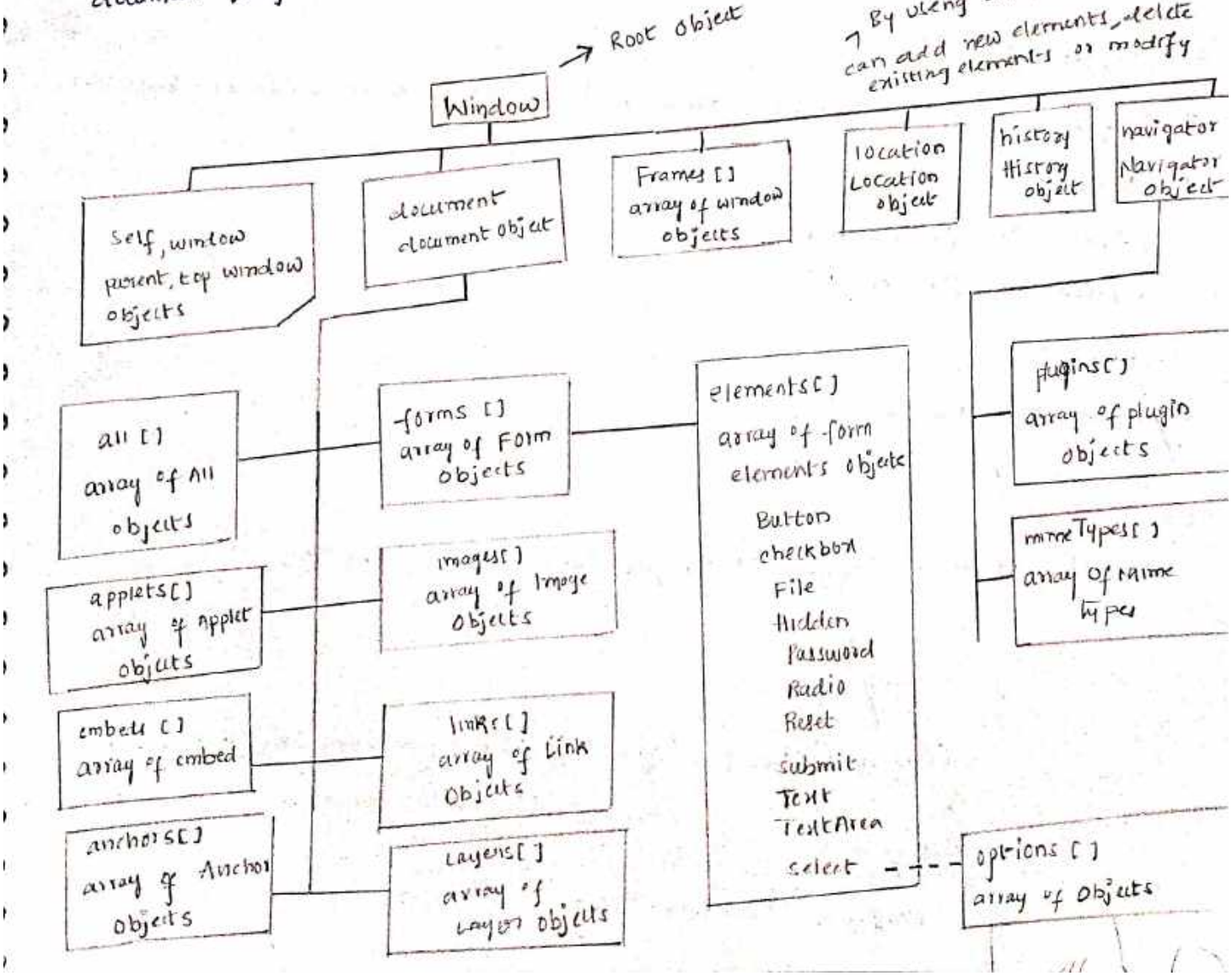
document.querySelector(".className / #id / tagName").

6) QuerySelectorAll() (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 messege")
alert("Line 1 \n Line2");
alert(10+20); (or) alert(Expression);
```

Ex:-

```
alert("Form submitted succesfully!");
```

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 delete this ?");

if (result) {
    Confirm("Deleted...");    // when you click Ok, this will print,
}                             otherwise cancel, else 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 tap).
* Developer can log all messeges as output in console.
* Only for Debugging, which will see the how internally it works.

common methods

a) console.log() → Normal messeges
b) console.error() → error messeges (Red color)
c) console.warn() → Warnings (yellow color)
d) console.info() → info messeges
e) console.debug() → debugging messeges.

Syntax:

```
console.log("string | Expression");
```

* Console cannot take Markup or Complex Representation
* Console methods are only for design 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)
* It allows you to change content of webpage dynamically using javascript
* It can include HTML tags, not just Text.

Syntax:

Object.innnerHTML

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 clicks content changes from Hello to Welcom to javascript (in bold)

5) OuterHTML → Changes entire element including the tag

* OuterHTML is similar to innerHTML, but it will the Replace the existing marking 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> kuruva Thirumaleh </p>";
```

Output

`<p> Kuruva Thirumaleh </p>`

outerhtml → to Remove whole div tag and new `<p>` tag appears instead.
→ Orginal tag - doesnot exist anymore

Difference b/w innerHTML and OuterHTML

innerHTML	OuterHTML
Returns only the inside the content of an element	Return the entire element including the tag itself
Replace the only the content inside the element	Replace the element itself (whole tag + Content)
Ex: update text, image,	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.

7) 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:

 Hello World </p>

document.getElementById("demo").textContent; or .textContent = " Hey

output

Hey

output:

Hello World

Java Input Techniques:

1) prompt() — Input from browser

* prompt() method provides a popup that allows input from the Browser

(or)

* shows a popup box asking the user to type something from the browser

* Empty String [' '] : Ok click without value
* String Value : Ok clicks with value
* null : cancel clicks with or without value

Syntax:

prompt ("your message,");

Ex: prompt ("Enter Something");

prompt ("your message", "default text");

prompt ("Enter your name:", "Hari");

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 allows users to enter different types of data
- Every form input stores the user-entered data inside the its "value" property.

Ex:

<input type = "text" id = "name" value = "John">

document.getElementById("name").value.

3) Query String

* Querystring 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 (key, value)

* 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("=")).substring(9); -from 9 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.

Ex: x=10; ✓

"use strict"
x = 10; ✗

instead we can write

"use strict"
Var x = 10; ✓

. declaration
Var x;

. Assignment
x=10;

. intilliasation
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 blocks inside the function
* It allows declaring, assignment, intilliazation.
* It allow shadowing → process of Re-declaring or Re-intilliasing with same name identifier within the scope.

Ex:
```
function f1()
{
var x;      → declaring
var x=10;   → intilliasing
if (x==10)
 {
  var y=20;
  y=30;     → shadowing
 }
document.write("x"=+x+"<br>"+"y"=+y);
}
```

* It allowls hoisting:

↓ process of the declaring and a using a variable anywhere. there is no order dependency. you can use and declare later in function

Syntax:

```
"use strict"
x=10;
document.write("x"+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, intilliazation.
* 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 Intilliazation
* It will not support declaring and assigning.
* It will not support shadowing, hoisting.

Variable Global scope

- Global scope defines the accebility of variable across the multiple Functions.
- Declaring the or Intialliing variable outside function makes as Global
- you can configure a global variable inside the function using "window" object Is.

syntax:

```
window.varName = 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'=*+'y"-y+"z"=+z+'a=+a);
}
f1();
```

when we cannot write object only we can only f1() only.

When we write window object here a value can accessed function f2() also

Variable Naming

* Name must start with an alphabet
* It can be alpha numeric with "_";
* you can start name with "_". But it is not Recommended.
* ECMA doesnot Recomended to use special chars.
* Name max length can upto 255 chars.
* 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 type Language
* Hence, there is no datatype Restriction for data in memory

var x = any type of data can be stored.

* Javascript is implicitly type Language. the data type changes according to value type.

Javascript datatype in memory can be

1) primitive datatype

* primitive datatype are immutable type
  ↓
  you cannot change the structure
* It have 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 deumal places]
exponent	$2e^3$ [2 x 10^3] = 2000
hexa	000fx45
octa	00578
binary	0b1010

f) Bigint

Bigint is a new data type for safely using large integers.

Bigint configures large integer values with "n" as suffix.

Syntax: var x = 9988+776543; // It is not safely.

var x = 9988877645 3n; // It is safe

b) String

* string is a sequence of characters used to Represent the text
* strings can be created with single qoutes ('...'), double qoutes ("...") or backticks (`...`)

Ex:- let name = "thiru"; or let name = new string ("thiru");

* 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) paseInt — 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 'is NaN()'.

FAQ) How to verify the input value is number or not using Regular Expression

- By using pattern and javascript "match()"

syntax:

```
Var age = "30";
Vare 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) toLocalString() - Converts the a number or date into a string based the User's Local settings (country format).

Java script String Type

*) String is a literal with group of characters enclosed in

a) double quotes " " } used to toggle outer and inner string

b) single quotes ' ' }

c) back ticks ` `

Syntax:

```
Var link = "<a href='home.html'>Home</a>";
Var link = '<a href="home.html">Home</a>';
```

* Binding versions of javascript requires a lot of concat techniques

Using "+" operator

Syntax:

"String" + dynamicValue + "String"

* ES1 versions of javascript introduced a data binding expression "${ }".

which you can embed in a string with backticks only.

Syntax:

`String ${} dynamicvalue ${} String`

backticks

* 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 formatting (or) building a string with a specific style presentation and manipulating string

↓

Modification existing (or) change string to produce a new string (or) change text content

* Javascript string formating includes

a) bold()
b) italics()
c) fontsize()
d) fontcolor()
e) sup()
d) sub()
e) strike()
f) toUppercase()
g) toLowerCase() etc

Syntax:

Var str = "Kuruva Thirumalesh";

document.write(str.bold());

document.write(str.bold().italics().fontsize(4).fontColour("Green"));

* Javascript - strings manipulation:

a) toUpperCase()
b) toLowerCase()
c) Length()
d) slice()
e) substr()
f) replace()
g) trim()
h) Split()
i) Concat() etc..

Events

① Onblur → Onblur triggers when user leave the input field specify (or) actions to perform when element lost focus

Example flow: user click inside textbox → Type something → click outside/ move another field → Onblur Runs.

② Onkeyup → onkeyup trigger everytime the user Release a key specify (or) actions to perform when a Key is Related.

Example: Flow: press Key → Key down → Onkeyup Run:

Note:

You cannot apply font and style to Rc data Elements using string methods.

Syntax:

document.getElementById("textBox-Id").value = "John.bold(); // Invalid.

* Javascript provides various methods for string manipulations

1) 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 a 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 = 95, 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 chars()

5) endsWith() : It Returns boolean true if string ends with specifics chars()

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 Return the first occurance 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 occurance 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 Start index.

10) substring() : It can extract chars between Specified index bidirectional.

FAQ)

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); => 7 to 14 index

str.slice(7); => 7 to end

str.slice(7,0); => 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); scrip

starts with s

ends before t because (-1) (end not included)

* Substr() Refers to the number of Characters from the specified index
* Doesnot allow negative values

Syntax:

str.substr(startIndex, howMany Characters (or) length);

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 index

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 treas as substring(0,4); // thiru

2) if startIndex > endIndex → javascript swaps them automatically

"thiru".substring(6,2);

↳ substring(2,6);

11) Split() : It can split the string into an array by using specrfice delimeter (or) seperator

Syntax: string.split(seperator, limit)

Var details = "John-23, david-25, sam-31";

details.split(','); // [John-23 (0), david-25 (1), sem-31 (2)]

"one two three".split(" ", 2); // ["one", "two"]

1) " " Split by space (or) Break sentence into words "Hello world" → ["Hello", "world"]

2) "," split by camma "a,b,c" → ["a", "b", "c"]

3) "-" Split by dash "2024-10-21" → ["2024", "10", "21"]

4) "" Split every character "Hi" → ["H", "i"].

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 = "99999";
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

14) replace() : Returns only the first matching substring with another value

Syntax:

string.replace(searchvalue, new value);

Ex:- msg = "Javascript is a language";
msg.replace("Javascript", "Java"); // Java is a language

15) repeat() : Repeat the string multiple times.

Syntax: string.repeat(count); Ex: hi.repeat(2); // hihi

16) includes() : Check if a string contains a given substring

Syntax : string.includes(searchValue); Ex: I am thiru.includes("am"); // True

17) padstart() / padEnd() : Add characters to the Begining of a string until it reaches given length.

Syntax string.padstart(targetLength, padstring) 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

true + true = 2
true + "B" = trueB
true + 10 = 11
true - true = 0

Undefined

* It Refers to a memory where is value 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

Ex: var x;
document.write('x= ${x}
 y= ${y}');

x = undefined
y = not-defined.

Null

* Null is a type configured memory Reference when value is not providing during the Runtime.
* It is verified by using "Null" keyword.

Ex: let a = null;
console.log(a); // null.

Symbol

* It is configured a hidden key for object
* hidden key are ignored over iterations. But they are accessible individually.
* Symbol is a new datatype of a javascript for developers from version ES6

syntax:

var key = Symbol();
var obj = {
[key]: value
"Name": "Thiru";
}

obj[key] ⇒ can access the value stored in key Reference

Non-primitive Data types

* They are mutable types
- They dont 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

a) Array → starts from 0 up to the n-1

Array is a collection of values in a sequential order, which you can access Randomly.

* Arrays are introduced into computer programming to Reduce overhead and Complexity.
* Arrays can Reduce overhead by storing values in sequential order.
* Arrays Reduce Complexity by storing the multiple values under the Reference of single Name.
* Arrays can stores 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 of 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) Intilliase the Array

a) [] → Static memory [the memory allocated for first Refernce will continue across multiple Requests]
b) Array() → Dynamic Memory [Which is a newly allocated everytime when Requested]

Syntax: Initialization
Var arrayName = [];
Var arrayName = new Array();

Assignment:
Var arrayName;
arrayName = [];
(or)
arrayName = new Array();

Ex: let name = ["thiru", 'shiva', 'bhaskar'];

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
b) Non-primitive
c) Function

Ex): Var values = [];
values[0] = 1; values[1] = "thiru";
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]()); // Hello Array.

[4][] = IIFE (Immediately Invoking Function Expression.

=> Function without name is called Anonymous Function

Ex 2: values = [[[10,20,30],[40,50,60]],[["A","B"],["C","D"]]]

[0] [1] [2] [0] [1] [2] [0] [1] [0] [1]

[0] [1]

document.write(values[1][1][0]); // C

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 seperated with ",".

b) join() : It is similar to string but uses custom delimeter.

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 occurance 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 = [40000, 421000, 67000, 68000, 23000];`

```
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

1) Explicitly Iterators for Array:

a) for..in : It Reads all properties (or) Indexes of Array.

Syntax:

```
for (var property in collection)
{
}
```

Ex:- `var sales = [45000, 421000, 67000, 68000, 23000]`

```
for (var property in sales) {
    document.write (property); }
// 0
// 1
// 2
// 3
// 4
// 5
```

b) for..of : It Reads all values of Array

Syntax:

```
for (var values of collection)
{
}
```

Note: Destructing Array ***

Javascript ES6 Version introduced Array Destructing

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.toString()} <br>
```

output

```
id = 1
name : TV
cities : Delhi, Hyd
```

Dynamically Creating HTML Elements:

1) HTML Element can be created in Javascript by using DOM Method.

Syntax:

```
document.createElement("tokenName")
```

=> 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 = "cssclassName";
```

4) Add Element into page: You can add any dynamic element using the DOM method

Syntax:

```
parent.appendChild();
```

Ex: <figures> </figure>

```
document.querySelector("figure").appendChild(pic);
```

Presenting ArrayElements in HTML page

1) Use any Iterator for Reading elements from Array.
   map(), 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 Reffered 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); // thiru

Note: "dot" is member invoking Operator.

* JSON data kept is kept in seperate JSON file that have the extension ".json".
* to acces data frim JSON File Javascript provides a "fetch()" promise.
* promise is aspecial type of function that handle the 'async" methods.
* "Async" is unblocking technique; without blocking other Requests it can poocess your Request.
* Synchronous → code is executed linebyline in the order it appears. Each line waits for the previous one to fmish before Running.

Syntax:

```
fetch("url").then(function(Response){
    return Response.json();
}).then(function(data){
    ... you can use data ..
}).catch(function(ex){
    .... display exception ....
}).finally(function(){
    ... executes always ....
})
```

fetch() is a built-in Javascipt function used to:

* Request data from APIs/servers
* 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 NItation.

syntax:

"Key": "Value"

* JSON is a lightweight data format used for storing and exchanging data, especially between a client & server.

Ex:

```
{
  "name": "thiru",
  "age": 20,
  "gender": "male",
  "skills": ["java", "Html"]
}
```

* Key are always string (in double qoutes
* used for sending the data b/w Frontend and Backend
* Eeasy to write and Read
* Lighter and faster than XML and Language independent (used in java, Python.).

JSON.stringify() : to converts Objects → JSON string

JSON.parse() : to converse 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. Ex: this.name

* the members of object are accessible outside the object by using "person.age".

FAQ)

1) How to Read all keys from object?

By using Object.key(Objectname) (or) for in loop

2) 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 pairs to map Object

Syntax:

mapObj.set(key, value); Ex: Map.set("name", "Thiru");

2) get() : It Returns the value of specified key.

(or)

get(key) : Returns the value for given key.

Syntax:

~~mapObj.get(value);~~ mapObject.get(key); Ex: m.get("name"); // Thiru

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)

months starts from 0 to 11
(0 = January, 11 = 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");
```

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 integer value b/w 0 to 31 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.

1) toLocaleDateString()
* Return only the date
* Formated based on Country/Locale like India, US, UK etc

Syntax
```
date.toLocaleDateString(locale, options);
```
Ex:
```
now.toLocaleDateString("en-IN");  // 26/11/25
```

2) toLocaleTimeString()

* Returns only the time
* with hours, minutes, seconds in Local time formate

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 use 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)

5) setDate()

sets the Date (0-31)

6) setMonth()

sets Month (0-11)
Jan Dec

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 15th Day of Month

a.setMonth(0); // January

* the above functions are used to change Date and Time.

JavaScript Timer Events

1) setTimeout():

* setTimeOut() is used to executes 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 Method 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.trun() : Removes the decimal part and Returns integer only.

Ex: Math.trun(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 values of base to the power of exponent

Syntax : Math.pow(base, exponent).

Ex:- Month.pow(2,4); => $2^4 = 16$

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 (possitive);

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 trigonmetric 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*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).

== (Equals to) : 2==2
!= (Not Equal to) : 2!=3
> (Greater than) : 2>3
>= (Greater than or equal to) : 2≥3
< (Less than) : 3<2
<= (Less than or equal to) : 3≤2
=== (strictly equal to) : 2===2

FAQ) What is Difference between "==" and "===" ?

== : can Compare values of various types

=== : can compare only identical typical 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 ||) : 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 → false
false → true

5) Conditional (Ternary) operator:

shortcut for if-else

Syntax:

Condition ? ValueTrue : valueFase;

Ex:

```
let age = 20;
age >= 20 ? "Adult" : "Minor";
```

6) Special Operators

a) typeof Operator : It is used 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();
(or)
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 Opelator: 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 collects (packs) multiple arguements 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 masks = 45;
if. (masks >= 35) {
  "Pass";
 }
else {
  " Fail";
}
```

c) if..else if.. 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 true
}
else {
  // Code block to be executed if all pending Condition 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 easy cleaner and easier than writing multiple if.. else -- if statements

Syntax:

```
Switch (expression) {
Case value1:
  // Code to execute if expression == value1
    break;
Case value 2:
  // Code executes if expression == value2
    break;
  .....
  default;
  // Code if no case matches
    }
```

Ex:

```
let day = 3;
Switch (day) {
 Case 1 :        sunday
  Console.log ("~~monday~~");
    break;
 case 2:
   // Monday";
    break;
 case 3:
   "Tuesday";
    break;
 Case 4 :
   "wednesday";
```

```
    break;                    // Tuesday
  default:
    console.log("Invalid Day");
}
```

2) 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:

```
for (initialization; condition; increment/decrement)
  // Code to Run
}
```

① ② ③ ④

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);
```

→ semicolon mandatory on while.

Ex:

```
var i = 1
do {
  console.log(i);   // 1
  i++;
}
while (i >= 5);
```

d) for ... in loop

* Used to iterate over object properties.

Syntax:

```
for (var key in object)
{
 // code
}
```

Var student = { name: "thiru", age: 20 }

Ex:

```
for (var key in student)
{
Console.log (student[key]);
}.   // name
     age
```

e) for ... of LOOP

* Used to iterative 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 executes 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

1) 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() {
  Consbe log ("Hello Javascipt");
}
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;
};
```

(or)

```
var addition = (a,b) => a+b;
```

Function Recursion:

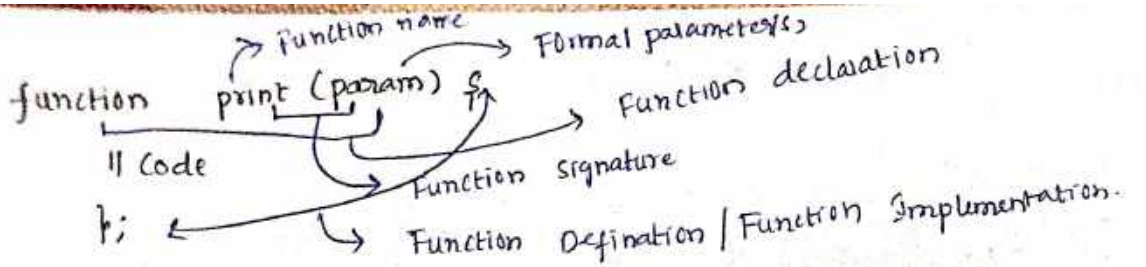
Function is a technique where a function calls itself until a specific condition is met

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

a) primitive
b) Non - primitive
c) 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 - seperated inside the parantheses

Syntax:

```
function functionName (Param1, param2, Param3 . . )
        {
        // code
        }
```

* Every parameter is Required Order dependency

Ex: function details (name, age);

```
{
}
details ("thiru", 20); ✓
detail (20, "thiru"); X
```

* If you want to ignore any specific parameter, then you have to handle using Undefined type.

* ECMA allows max 1024 parameters.
* ES5+ versions introduced "rest" parameters.

Rest Parameters (... param)

* Define using ...paramName
* Rest Parameter is always of array type
* A Function can have only one rest parameter.
* Rest parameter must be the last parameter in the function

Syntax:

```
function demo (... param1);
```

Ex:

```
function sum (... nums) {
  return a+b;
}
sum (10, 20);
```

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:

*) Rest Parameters is about formal parameters, a single formal Parameter allow multiple arguments

* Spread parameters is about, a single actual parameter can spread into multiple formal parameters.

Function Promise

* A Promise is an object that handles asynchronous operations.
* It Represent the value that will be available now, later, never

1) pending
2) Resolved (fulfilled) = Defines then()
3) Rejected ⇒ Defines catch().

* Asyn = Is an unblocking technique that allows the task to perform without blocking other tasks.

Syntax:

```
var refName = new promise (function (resolve, reject) {
    if (condition) {
    resolve "success";
    }
    else {
     rejected "failure";
    }
});
refName.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

18-11-25

* 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:-

```
le x = y+10;
```

y is not defined → ReferenceError

What is Exception ?

* An Exception is a Runtime event that distubs 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.
* 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 exceptions that occur in the try block.

Syntax:

```
try {
  // Code that might thrown 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 occured 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 handler the error
   }
finally {
    // code always executes (cleanup)
     }
```

Javascript OOPs.

* OOPs = Object Oriented Programming System

* Features (Advantages)
  - Code Reusability
  - Code Seperation
  - Code Extensibility
  - Code Security

* Drawbacks
  - No direct low-level memory access.
  - Cannot interact with hardware directly.
  - Takes more memory
  - Complex Configuration
  - Testing for small tasks.

* Evolution of OOP

1960 - Alan Kay introduced the idea of 'objects' in programming

1967 - John Olay & Kristian Nygard 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, function, 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:

* planned export

```
export const name = "John";
export function greet() {
    return "Hello";
}
export class Student {
}
```

* 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";
```

* Name exports use { }
* Default exports does not use { }.

Ex: `import welcome, {greet} from "../home-module-js";`

Classes :

* A class is a template provides sample data and logic which you can customise and implement according to requirements

(or)

* A class is a template or Blueprint used to Create Objects.

* Class Contains properties (data), methods (functions).

1) Class Declaration

```
class className {
    // properties
    // Methods
}
```

2) 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.

FAQ:

1) Can we define variable or function as class member?

No

2) Can we define a variable or function in class?

Yes

3) Why function and variables are not allowed as class members?

Because, ~~A class member~~ variable and function are immutable.

A class member are always must be mutable.

hence, Variable and function are not class Member.

a) property

* property is a key-value pair attached to an Object
* properties are variables (data/values) that belongs to Object or class Instance
* property is a mutable which we can change the Behavious dynamically.

Ex: let product = {

```
name: "Rahul";
age: 25;
city: "Kurnool".
};
```

// properties.

b) Accessors

* Accessors are to used to fine grained control over the property

* Accessors = Getters + setters

1) Getter
To Read property
get propertyname() {
    return value
}

2) setter
to write/update property
set propertyName(newValue) {
    // validation
}

c) Method

*A Method is a function that is a property of an object or class.

* Methods define Behaviour for Objects

* Methods are Mutable

* They can access Objects properties using 'this', and are called with parentheses ().

Syntax:

class Method:

```
Class ClassName {
    methodName(params)
    {
    }
};
```

Object Method

```
let ObjectName = {
    property1 : value1,
    Property2 : value 2,
    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 in side the class

Syntax:

```
Class ClassName {
constructor (param1, param2, ....) {
    this.parameter1 = parameter1;
    this.parameter2 = parameter2;
    } }
```

let nam= new ClassName (param1, param2, . .); // Object Creation

* Only one constructor method is allowed in a class.

**

Code Reusability and Extensiability :

We can handle the code Reusability and ~~Reasabi~~ Extensability by using 2 technique.

a) Inheritance

* Iheritance 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 (child/subclass) inherits properties and methods from another class (parent/superclass/Base class) (derived class.)

* Represents "Is-A" Relationship.

* Code Reusability

Syntax:

```
class ClassName {
  // Properties & methods
}
class subclass extends superclassName {
    // properties & methods
  }
```

** extend keyword

* extend is a keywod, used for a class to inherit properties and methods from another class.

** super keyword.

* super keyword is used in a child class to Reffer the parent class.

* call parent constructor

* Call parent methods

b) Aggretion

* 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.

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 make sound");
    }
}
class cat extends Animal {
    speak() {
    console.log("cat makes sound");
    }
}
Const animal = [new Animal(), new Dog(), new cat()];
    animal.speak();
// here same speak method name behaves differently dependending on the object type
                                                    that is the polymorphism
```

Event Handling in Javascript

What is Event ?

- An event is a Message / notification send by a sender (HTML element) to a subscriber (Javascript Function) when something happens on the web page

Ex:- click, keypress, mouseover etc.

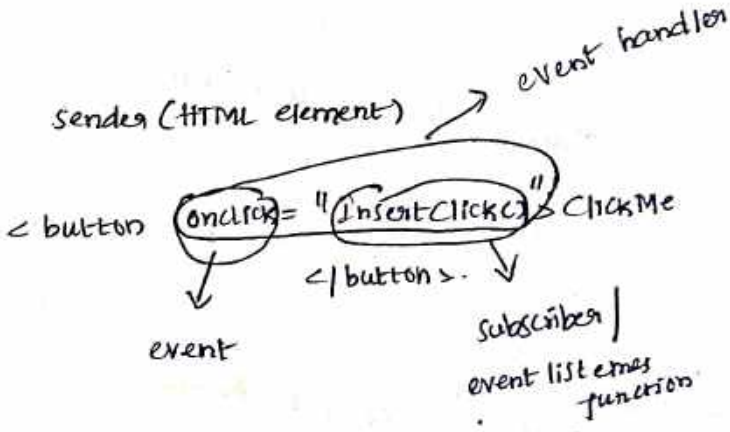
* Javascript event handling follows a software design pattern called Observer Pattern.
  - sender (subject) : Elements triggers events
  - subscriber (Observer) : Function that listeners for the event.

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) ondbclick : Triggered When the user duble-clicks an element

syntax:

ondbclick = "functionName()"

c) onmouseOver: Trigger when the mouse pointer moves over an element

syntax:

onmouseover = "functionName()"

d) onmouseout : Trigger when the mouse pointer moves out of an element

syntax:

onmouseout = "functionName()".

e) onmousemove: Triggered continously 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 : Trigger when the mouse button is Realeased after pressing

syntax:

onmouseup = "function()"

10) oncontextmenu : Fires when the Right-mouse button clicked on an element

2) 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 : Fires, when the value of an element changes and the element lossesfocus

Syntax

<element onchange = "functionName (this, event)"></element>

b) Onblur : Fires, when an element losses focus.

syntax:

Onblur = "functionName (this, event)";

c) Onfocus : * Fres, 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) onpaste : triggers when the pastes 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 executes a function only once after a specified time

Syntax:

setTimeout (function() {}, interval).

b) clearTimeout() : to stop/cancel a sheduled setTimeout(). before it Runs

Syntax:

clearTimeout (TimeoutReference);

c) setInterval() : executes a functionality at Regular time intervals.

Syntax:

setInterval (function() {}, Interval);

d) clearInterval() : to stop a Running setInterval()

syntax:

clearInterval (IntervalReference);

7) scroll events → When a page or an element is moved up, down, left, Right.

Onscroll : Fires 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 seperation of Code and UI.

Syntax:

element.addEventListener(event, function() {});

Note:
event write in addEventListener like below:
"onClick" ✗
"click" ✓

* Removing Event Listeners

syntax

Element.removeEventListener (eventName, functionName);

Ex:-

```
document.getElementById("btn").addEventListener("click", function() {
        alert('Button Clicked');
    }
);
```

FAQ) how to change events for elements dynamically ?

By adding event Listner.

State Management in Web Applications.

* Every application uses Http or Https as a protocol

* Http is a stateless

  Stateless → Cannot Remember information pages or Requests.

* Http – Hyper Text Transfer Protocol
  - used for transfering 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)

techiques:

a) Query string

* Data is appended in the URL with a key-value formate.

? key = value

* Access: location.search

b) Local storage

* Local storage is ~~permanent~~ permanent storage.
* Local storage is available even after your device shut down.
* Local storage is accessible all ~~the~~ 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 = thiru"; 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 29:20:40 ";

Attributes

expires: specify the exact date/time when the cookies should expire

max-ag: specify the lifetime of the cookie in seconds

path : defines the URL path where the cookies is accessible

Note:

* LocalStorage and session Storage 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 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.

1) 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) getElementB~~

d) QuerySelector → selects the first element that matches a css selector

e) QuerySelectorAll → 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) SetAttoibute(): 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.removeEventListner(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 a Object Representing the Future Result of an asynchrouns operation, which can either success (Resolve) or fail (reject)

(OR)

* A Promise is an Object Representing the eventual completion (success (OR) resolve) or failure (Reject) of an asynchronous operation.

* A promises will helps to handle the asynchrous code.

* A promis Can be 3 states

a) Pending : intail states, task not completed yet

b) FullFilled [Resolve] : task Completed Succesfully.

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 handles the Resolved (fullfilled) value of a promise

* It Runs only if the promise is successfull (ie; resolve() called).

* Handles the success.

- catch():

* catch() is used to handle the errors when promise is Rejected.

* It rens 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 used for clean up tasks.

syntax:

```
myPromise
    .then(result) =>
    {
      // handle success
    }
    .catch(error) =>
    {
      // handle failure / error
    }
    .finally(() =>
    {
      // 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 untill the promis is Resolve or Reject.

Ex:

```
async function getUserdata() {

    try {

     let response = await fetch ('https://jsonplaceholder.typincode.com/users/1');


     let data = await response.json();

      Console.log('User Data?:', 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) Call backs
2) promises
3) Async/Await (modern way (Recommonded)) . ✓

Set

* Set is the one of Datatype in javascript
* A Set is 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);
```

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 instragram, whatsapp, Facebook, Spotify, Netflix, pinterest etc; uses React, for creating single-page Application (SPA).

* React and React.js are bolh 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 Architechure
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) Reausability.

} Core Features.

What is Reconciliation? → Reconciliation is the process by which Reacts updates the Real Dom by comparing old and new Virtual Dom using diffing algorithm.

Challenges / Issues / problem in React:

1) Frequent updates
2) Complex tooling
   * Setup can feel Complex.
3) Not a Complete Framework
4) Performance Issues
5) Javascript Depedency
   * If javascript is disabled, React app wont work properly

FAQ:

1) What is Virtual DOM? → It is a Duplicate Copy of actual dom
* 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 comparing 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 Feacture that allows you to create a seperate DOM and css encapsulation for Components, do not affect the Rest of the page
(or) shadow Dom is a hierachy 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 Reccommended.
* Supports Javascript inside { }
* Browser does not understand Jsx
* Babel Converts Jsx to Javascript

Ex: React with Jsx (Reccomended)

```
function App() {
  return (
    <div>
      <h1> Welcome React</h2>
      <p> Hello</p>
    </div>
    );
}
export default App;
```

{ Clean & short, Easy to Read }

React without Jsx (Pure React)

```
function App() {
return React.createElement(
'div', null,
React.createElement("h1", null, "welcome
  React"),
React.createElement("p", null, 'Hello')
);
}
export App;
```

{ Hard to Read, to much nesting, Difficult maintaining }

Jsx Rules :

1) All tags must be closed

Ex: <img src="photo.png"> ✗ ✓

2) Jsx is case-sensitive
   * Components should be uppercase
   * Html elements in Lowercase

Ex: <MyComponent />

3) Javascript must be inside { }

Ex:
```
function App() {
  var name = "thirumalesh";
  return ( <h1> Hello {name} </h1>); }
```

4) Use className instead of class.

Ex:

```
<div className = "box" ></div>
```

5) Use htmlFor instead of for

Ex:

```
<label htmlFor = "email"> Email </label>
```

6) Jsx will not allow individual elements in ui.

Jsx requires a container (or) one parent Element/component or fragments to handle elements.

Ex:

```
<h1>
 Hello
</h1>
```
=) invalid

```
<div>
 <h1>
  Hello
 </h1>
</div>
```
=) valid

7) Jsx, Event handles use CamelCase

```
<button onclick = {handleClick} > Click </button>   // invalid
                                                      (onclick)
<button onclick = {handleclick} > Click </button>   // invalid.
```

8) Inline styles uses Javascript Objects

```
<div style = {{ color: "red", background-color: "blue" }} ></div>.
```

9) Jsx comments use {/* */}

10) Boolean & null values are not rendered

```
{true}
{false}
{null}
```
// invalid

FAQ)

What is Babel ? → Converts Jsx into Javascript

Babel is a Javascript Compiler that converts [redacted] or Jsx code into plain (or) backward compatible javascript that the browser can understand.

* Here, Browser doesnot understand the Jsx (or) modern Javascript (Es6+) then we will use babel compiler for to converts the Jsx into plain Javascript for Run the code in browser.

Ex: Jsx code

```
const name = "Thirumalesh";
Const element = <h1> hello {name} </h1>
```

* the above code Babel Converts Jsx into Javascript that browser can understand.

```
Const name = "thirumalesh";
const element = React.createElement(
                "h1", null, "Hello", name );
```

FAQ)

1) 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 (CCRA)

* It is an official tool of React which setup a new project with all neccessary configurations automatically.
* pre configured webpack & Babel
* Supports JSX and ES6+ automatically

how to create a React App with CRA

npx create-react-App Folder(or) projectname

Ex:- npx create-react-app my first project

start the development server

npm start

↳ opens the application in browser : http://localhost:3000 → default port in React.

CRA Folder structure :

My-app
- node-modules
- public/
  - index.html
  - favicon.ico
  - manifest.json
- src/
  - App.css
  - App.js (Root component)
  - index.js (or) main.js [Entry point]
  - index.css
  - logo.svg
  - reportWebVitals.js
- .gitignore
- package.json
- package-lock.json
- README.md

Flow of React

Browser
↓
index.html (load once)
↓
div # root found
↓
index.js loads
↓
App component rendered
↓
Child components
↓
JSX
↓
Virtual DOM updates
↓
Real DOM update.

Diffing Algorithm (Compare old virtual dom vs new virtual dom)
↓
Real Dom

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 including 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.

(or)

- index.html loads only once
- Whole page doesnot Reload.
- Only the required React Components change (or) Re-Renders dynamically.

* Content Changes dynamically

Components in React

* Components must be extension with .js or .jsx : (File name)
* Components are the building blocks of the UI.
* Component is a Reusable piece of UI.
* A Component comprises 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

1) Functional Component (Recommended)

Syntax:

```
function ComponentName() {
    return (
        <div>
        Jsx UI
        </div>
    );
}
export default ComponentName();
```

(Ex):

```
function Welcome() {
    return (
        <div>
        <h1> Welcome to React </h1>
    }
    export default Welcome;
```

```
import Welcome from "./welcome";
function App() {
    return (
        <div>
        <Welcome/>
        </div> ); }
```

Ex: Header, footer, Main, Nav, Section ble can beild by using components.

2) Class Components (Legacy or Not Reccommended)

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>
<h>Welcome to React</h>
    </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 created a file with .css extension and import into your Component.

Ex:- .title {
  color: blue
  font-size: 24px;
  }

2) Inline css (styles Attribute)

*In this, Jsx uses Javascript Objects for styles.

Ex:

```
function App() {
 return (
  <h1 style = {{color: "red", fontSize: "24px"}}></h1>
  );
 }
```

(or)

```
function App() {
    var mystyles = {
            {color: "red",
             fontSize: "24px"}
            }
     return (
      <h1 style = {mystyles}></h1>
       );
      }
```

Note:

the below Rules applies to inline styles Only.

*) css propety Name must follows camelCase

Ex: backgroundColour, fontSize, textAlign, marginTop.

3) css Module (Reccomended)

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;
  }

```
import styles from "./Button.module.css";

function Button()
{
return
  (
  <button className={styles.button}>Click Me</button>
  );
}
export default Button;
```

* If multiple classes

```
.card {
    border: 1px solid gray
}
.highlight {
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 changed
* props like asguments to a function

* Ex: Child component

```
export function StudentCard (props(or)p){
 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={1}
    name="thiru"
    marks={85}
/>
```

Note:

We cannot modify props inside a child component. (or) props can only Read, not changed.

Ex:- `props.name = "John";`

```
name = "thiru";
id = {1};
marks = {85}
```

Props in Class Components:

{this.props.name} (or) Syntax: {this.props.propsNames}

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:

```
<React.Fragment>          * we can pass the key.
    <h1>Hello</h1>
    <p>Welcome to React</p>
</React.Fragment>
```

(or)

```
<>                        * cannot accept key
<h1>Hello</h1>
<p>Welcome to React</p>
</>
```

<> </> → 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

*) attributes (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 Feautures, also called as in-built functions
* Hooks are introduced in the React V16.8 version
* Hooks are backward-compatible with Old Feautures 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()

FAQ?
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 from react functions only
2) only call hooks at the top level ie; dont 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 Compon
* When state changes, the Component re-renders and update data in UI

Syntax:

```
Const [ state , setState ] = useState( initivalue );
```

- state: current value of state
- setState: function to update state
- initivalue: 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" : 'thiru",
          age : 18,
        });

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>
  ); }
```

Usecase

1) Form handling
2) Button clicks
3) User interaction
4) Counter
5) toggle

2) UseEffect()

* UseEffect() is an one of the hook method.
* It is used for to perform the side Effects in your components
* It is use for a perform the side Effects after a Component (or) renders-

FAQ)

1) When we use useEffect() ?

* Fetching the data from API (3rd parties)
* Updating the dom manually
* Cleanup operations (setting timers (setTimeOut, setInterval)).

Syntax:

```
useEffect ( () => {
              // Code to Run (side Effect)
            }, [depedencies]);
```

1st arguement:

this is a function that Runs your side Effect

-) fetching data from API
-) update dom manually
-) Cleanup Operations.

2nd arguement

optional array called depedency array.

- Controlls when the Effect function [first arguement] should run or

  the run.

how it will work ?

- In Class Components, we have lifecycle methods like ComponentDidMount, ComponentDidUpdate, ComponentWillMount.
- In fonctional Components, UseEffect() can handle all three methods, depedending on the second arguement (depedency array).

* useEffect() will depends 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 mount

```
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) => (
                <li> {us.user} </li>
                )) }
        ); } </>
```

2) Component DidUpdate ()

* It is a life cycle method in class Components
* It runs after Component updates (ie, after, state or props change)
* Runs when state or props change
* When the props or states change the useEffect will call
* Simply, when values changes, Runs logic (re-render)

Syntax:

```
useEffect ( () => {

}, [dep1, dep2 etc] );
```

Usecase:

* 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( () => {

    // cleanup function
    return () => {

    };

});
```

② If second argument will have a property or state in array then depend upon that particular value update the useEffect will call.

① 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 use effect will call.

Ex: Cleanup operation

Ex:

```
const [count, setCount] = useState(0);

useEffect( () => {

    let timer = setInterval( () => {

        setCount(count+1)

    }, 2000)

    return () => clearInterval(timer)

}, [] );
```

FAQ)
1) What happens if you dont pass a second argument (no depedancy 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 depedency array?

the effect runs after first render.
. then it runs again whever any value in the depedancy 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 Seperation
3) Better Readability
4) Easy to maintain and test

When should you create a custom hooks?

* 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
1) Return JSX	1) Name starts with use
2) Responsible for UI rendering	2) Responsible for only logic, not UI
3) Handles Html elements, lists, tables, dropdown etc.	3) Returns data + functions
4) UI	4) Reusable across multiple Components
5) Face	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 ?

1) OneWay - Binding :

* Data flows in One direction (uni-directional)
* data flow from Js → Html
* React follows One-way Binding by default.

Ex:
```
Const [name, setName] = useState ("Ramu");

return (
    <div>
    <h2> {name} </h2>

  < button onClick = {() => {
                    setName ("shiva");
                  }}>
          change </button>

    </div>
  );
```

What is Data Binding ?

* Data Binding means Connecting data (state) with the UI, when data (or) props(s) variables changes, the UI automatically updates
* states, props, variables, expressions inside JSX, must use in this { }

2) Two-way Binding (using Onchange) ✓

* Data flows in both directions
* Data flows from Js ⟷ Html which means Js to html and from html back to Js.
* Two-way achieved by combining value and Onchange handlers

Ex:
```
Const [name, setName] = Usestate ("Ram");

return (
    <div>
    <h2> {name} </h2>
```

```
<input type = "text"  value = {name}  onChange = {() =>
                                    setName(e.target.value)} />

        </div>
      );
```

What is Props Drilling ?

the concept of passing data from parent Component to all its child Components that means at every level components - using props. this called props 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.

** If we are passing data from 1st Component to 200 component, in the middle if any one of component not working then entire communication will fail.

* Un-neccessary data has to pass to every level of Components.

props drilling Issue →

to solve those problems / Issues we have to use

01) useContext() (or) Context API

02) Redux architechure.

03) Component Composition

What is Context?

* Context is a one of the Top level API in ReactJs

* 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)

1) 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.

Q) How to create Context in your React Application ?

- Using React.createContext()

```
Consr Context = React.createContext();
```

- React.createContext() will gives

Context.Provider → it will create Global store pass the data.

Context.Consumer → It is used consume data of a provider

Q) 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} >
  < Component200/>
</Context.Provider>
```

Ex: `< Context.Provider value={'third'} >` `< Component200 />` `</Context.provider>`

Q) What is Contex.Consumer ?

Context.Consumer — it is used Consume data of provider

```
< Context.Consumer >
  {value}
</Context.Consumer>
```

(Old/Traditional way)

```
<MyContext.Consumer>
</myContext.Consumer>
```

Syntax

```
< myContext.Provider >
  value = {user}
</myContext.Provider>
```

→ Provider

(or)

Recommend (Modern way) ✓

```
Const user = useContext(MyContext);
```

✓

Consumer

What is useContext() ?

* useContext() is a hook method
* Return context value
* Which is used to consume data in functional component
* Its 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: "Thiru";
    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 increases
* It can be used to access a DOM elements directly

(or)

* 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 can be update without re-rendering

Syntax:

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";

current.name = "thiru"; → updating the data

FAQ)

1) Where / when do we useRef()?

Re-Rendering will not happen

accessing DOM element

OnFocus - input field

render c) - 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-Render

```
Const count = useRef();
```

useRef() : returns an Object,
It is used to access DOM elements directly but it doesnot cause components to Re-Render

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 state logic in functional components.

* Return an array.

FAQ)

1) What is a Reducer in (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.

ie; (previousState, action) => newState

Previousstate State (intailState)

↓

IntialState

Ex: Const intialState = {Count:0}

Reducer Function Syntax:

function reducer (state = intialState, action)

{

return newstate;

}

2) What is an Action ?

An Action is a plain Javascript object that describes what happened.

structure of Action:

```
{
  type : "INCREMENT",

    Payload : {
       name : "thiru",
       age : 20;
       count : 1
       }
    }
```

3) What is dispatch?

dispatch is a function used to send an action to the user.

Ex: `<button onClick={() => dispatch ('increment')}> add </button>`

`<button onClick={() => dispatch ({type: 'increate'})} > add </button>`

`<button onClick = {() => dispatch (actionCreator)} > add </button>`

4) What is useReducer()?

* useReducer is a React hook
* similar to useState() but it is used for complex state managment
* 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.

5) Why we use UseReducer()?

* state logic is complex
* Complex logic

6) How does useReducer() work ?

Component loads with intial state

dispatch(action) is called

React call reducer(intial state, action)

Reducer returns new state

Component re-renders with update

useReducer Syntax:

```
Const [ state, dispatch ] = useReducer (reducer, initialState);
```

reducer → function to update

initialState → starting state

state → current state

dispatch → function to send actions

Work flow:

InitialState → ui shows data

↓

User clicks button

↓

dispatch (action)

↓

reducer (state, action)

↓

Switch (action.type) ← called React automatically

↓

new State returned

↓

React updates state

↓

UI re-renders with new data

1) First, initialState data is shown in the UI
2) When user click button, dispatch sends an action.
3) Reducer call by react automatically
4) Reducer checks condition (action.type)
5) Matching case runs and returns new state.
6) React update the state (in state)
7) Updated state is display in the UI

* 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)
setCount(count+1);

2) Logic inside the Component

onClick={()=>setCount(count+1)}

3) Good for simple state, single values

useReducer

1) You sends an action (action-based approach)
dispatch({type: "INCREMENT"})

2) Logic is outside the Component (in reducer)

function reducer(state, action)
{
// Logic
}

3) 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 Hooks that memorization (caches) the result of a Calculation and 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 re-render, all Calculations run again

* Heavy Calculations can slow down the app.

* by using, useMemo() avoids neccessary re-calculations.

Syntax:

```
const memoizedValue = useMemo(() => {
        return calculation;
    }, [dependencies]);
```

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 components 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-render:{count} </button>
    </>
    );
}
```

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 memorized Version of a Component that only changes when its dependencies change

Syntax:

```
Const memoizedFn = useCallback ( () =>
                    {
                    // Logic
                    }, [dependencies]);
```

---

rfc / rfce / rafce (vscode snippets in -functional Components)

1) rfce + Enter → React Functional Component with export

2) rfc + Enter → React functional Component (no export, default).

3) rafce + 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:

```
{ status === "loading"
  ? "loadig...."
  : 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 ext.

Event Rules in React:

1) Event names are written in camelCase

Ex: onChange, onSubmit, onClick etc

2) pass the Function Reference, not Functional call.

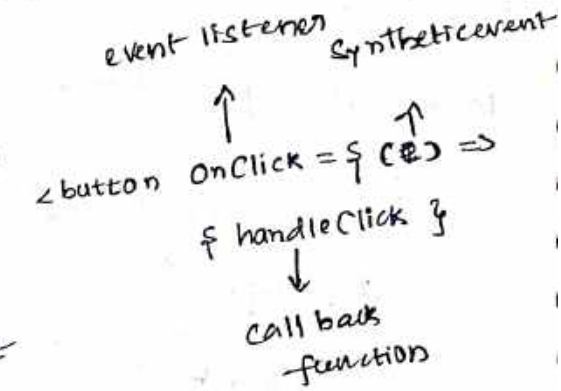
Ex: onClick = {handleClick()} ✗

onClick = {handleClick} ✓

Synthetic event

React event object (e) passed into your function

Ex:



Event Object has properties like:

event.target → element that triggered the element

event.preventDefault() → stops the Behaviour

event.type → type of event

Ex:

```
function ClickhandleC()
{
alert("Button clicked!");
};
return (
<>
<button onclick={handleclick}> click here </button>
</>
```

(OR)

```
function etc()
<button onClick={() =>
alert("Button click")
}>
Click here
</button>
```

1) Mouse Events
a) onClick
b) on Doubleclick
c) onMouseEnter
d) on MouseLeave
e) on Mouse Down
f) on Mouse Up
g) On Mouse Move
h) onContext Menu (Right-click)

2) keyboard Events
a) onkey Down
b) onkey Up
c) onkey UP

3) Form Events
a) Onchange
b) on Submit
c) On Focus
d) onInput
e) on Blur

4) clipboard Events
a) oncopy
b) onpaste
c) oncut

Dynamic Expression :

- dynamic expression is any Javascript expression written inside { } in jsx.
- React evaulates it at render time and shows the results in the UI.

Syntax:

{ expression }

Ex: let name = "thirumalesh";
<h5> Hey Hello {name} </h5>.

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), drop downs, tables, cards/components from API data

Syntax:

```
{
  array.map((item) => (
    <JSXElement/>
  ))
}
```

Ex:

```
Const names = [{id:1, name:"thiru"},
               {id:2, name:"shiva"}
              ].
return (<ul>
  {
  names.map(users) => (
    <li key={users-id}>
      {users.name}
    </li>
  ))}
</ul>.
```

FAQ)

What is a key!

*Key is a special attribute used ~~when~~ when 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
* , dropdowns (<select>), tables
* Javascript arrays + map() to create lists.

Arrays:

* Array are 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 : "thiru"   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 formate

Syntax:

```
Const users = [
    { name : "thiru",
      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 Routers

* React Router is used for Client side Routing in React SPA

1) What is the React Router Version?

V6 (latest version)

2) 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
* Routes is the Routing

Ex:

```
<BrowserRouter> {/* Router */}
  <Routes>  {/* Routing starts */}
    <Route path="/about" element={<About />} />
  </Routes>
</BrowserRouters>
```

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, thymleaf.

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) Routes :

* Routes 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 { Routes } from "react-router-dom";

<Routes>
  {/* Route Components */}
</Routes>
```

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>
  <Routes>
    <Route>
    <Route />
  </Routes>
</BrowserRouter>
```

Ex:

```
import React from "react";
import { BrowserRouter, Routes, 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.

(09)

* <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> tag is used to navigate from one page to another page by Reloading page

<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

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=react",
    hash = "#top",
  }}
> Search </Link>
```

2) NavLink (navigation + active styling)

* <NavLink> is a special version of <Link> in React Router that is used for navigation with active state.

* <NavLinks> 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: (or) Dynamic Navigation.

* navigate is used to move from one route to another using Javascript Logic instead of clicking event without page Reload.

* UseNavigate() is a hook that allows navigation between routes ^ 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 delā → 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 : "#",
    }
);
```

(OR)

```
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
/dashboard/settings
→ child routes

* To reuse Layout
* To avoiding Repeating header/Navbar.

Syntax:

```
<BrowserRouter>
  <Routes>
    <Route path="dashboard" element={<Dashboard/>}>
      <Route path="profile" element={<Profile/>}/>
      <Route path="settings" element={<Settings/>}/>
    </Route>
  </Routes>
</BrowserRouter>
```

<Outlet/> ⇒ placeholder Component
⇒ used in a parent Route to display its nested child routes.

```
function Dashboard() {
  return (
    <>
      <h2> Dashboard </h2>
      <Outlet/>
    </>
  );
}
```

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 Unauthorised access

404 → Page not found ,

401 → Unauthorised

403 → Forbidden

500 → server error

Syntax :

< Route path = "*" element = { <h2>404- Page Not Found</h2> } </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

"*" React Routers 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 means 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 :

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

5) Support RESTFul Design

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 are dynamic path in the URL that can change.

(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/>} />
```

static — dynamic.

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 params.

Syntax:

```
const Params = useParams();
```

* useParams() returns an object containg :
  - 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/23G31A0536

Return Object :

```
{
branch : "cse",
roll : "23G31A0536"
}
```

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 /: uage ?" element = {< user />} />

URLs that work:

/user/ thiru

/user/ thiru / 23

< Route path = "/user/: Uname ? /: uage ?" element = {< user />} />

URLs that work:

/user/ thiru / 21

/user/ thiru

/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 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.

/ 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 parameter (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
  - searchParams: used to Read Query params
  - setSearchParams: Function used to update Query params.

~~Important methods in URLSearchParams~~:

Important methods in URLSearchParams();

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 params exists.

4) delete (key): → to delete all keys with same keyname.
(or) delete (key, name); → to delete particular key

Removes a Query param Completely.

5) forEach (callback)

hoop 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 removing existing one (If it is already exists).
* Doesnot 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 = () => {
    SetSearch Params (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 javascript Web API that provides an easy way to Read, manipulate, and create Query parameters in the URL

* React uses it internally for useSearchParms.

* 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 = thiru #dev

```
Const location = useLocation();
Console.log(location);
```

Output:

```
{
pathName: "/user/21,
search: "? name = thiru",
state : null,
has : "#dev"
}
```

✓

Difference b/w path Params and Query Params ?

	Path Params	Query 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 useSearchParam()

5) Part of the URL path

6) @PathVariable

7) Identifies the Resource

Ex: /user/21 → tells which user

8) Who/what

after ?

@RequestParam

provides extra info or filters about that resource

Ex: /user/21 ? name = shiva & age = 21

Page and name give additional info.

How/which / optional info

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>

(or)

```
Const navigate = useNavigate();
navigate ("/user ? name = thiru ");
```

(or)

```
navigate ( {
pathName: "/user",
search : "? name = thiru"
});
```

3) Using URL Search Params / useSearch Params

What is search attribute ?

* Search attribute is part of URL that contains Query parameters.
* It starts with ? and includes key-value pairs seperated 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 intial Page Load (or) at once.

* Faster Intial 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 untill 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 untill the Component finishes loading.
* Messege 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 && (
    <Suspense  fallback={<h2>Loading .... </h2>}>
      </profile>
      </suspense>
    )}
   </>
   }
```

Flow:

Condition true :

- React tries to render </Profiles>
- 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 Resposible fo Rendering

2) React itself is Responsible for Rendering.

3) Once a lazy-loaded Component's Code is downloaded (js file stored in browser cache + memory), it is cached and Reused, It is not downloaded again.

4) lazy loading works with only dyaut exports.

5) When the page Refresh then chunk.js download it again

6) When the ~~Run~~ Re-Render the page then it will Reuse from memory.

Note:→ When the code download then only that code with Render/Run

Lazy Loading in React Routing:

Route Components are loaded/downloaded only when the user visit that Route

Why lazy load Routes?

* Without lazy loading:
  - All Routes → download at once → one big bundle.js
  - slower intial load
* With lazy loading:
  - Routes split into Seperate Chunks.
  - Pasten First page Load.
  - improve Performance and better for performance in large apps.

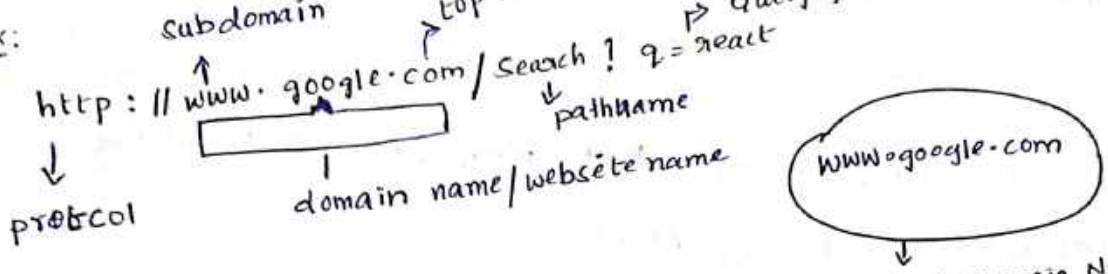
Syntax: →
```
const Profile = lazy (() => import ("./Profile"));
const Home = lazy (() => import ("./Home"));
<BrowserRouter>
  <Suspense fallback={<div> loading.... </div>}>
    <Routes>
      <Routes path="/Profile" element={<Profile>} />
      <Routes path="/Home" element={<Home>} />
    </Routes>
  </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:



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 ⟶ browser converts this into an IP address using DNS and connects to server.

Subdomain | domain | top level domain (India)

- Which saver you are Requesting

- .com → e-commerce websites
- .org → government sites
- .in → india country
- .us → USA country
- .edu → education Institutions.

3) Path
- Which ~~Server~~ resource you want from the server.

4) Query Parameters :
- Extra information / filters / sorts / pagination.

4) Port Number :

Port number that identifies which application should Recieve the Requests.

- default port : 80 (http)
default port : 443 (https) : amazon, meesho, flipcart, 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 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    methods, headers, body
```

How fetch() works ?

- Browser sends Request to server
  ↓
- Server Responds (JSON/text)
  ↓
- fetch() return Promise
  ↓
- Converts response using .json()
  ↓
- uses the data.

Ex: 1.

```
fetch ("https://jsonplaceholder.typicoder.com/users")
.then (response => response.json())
.then (data => { console.log(data);
})
.catch (erro) {
  Console.log ("Error", error);
  };
```

Ex: 2:

```
async function addData () {

try {
// Wait for the response before Continuing
Const response = await fetch("https://fakestoreapi.come/products,

  { method : "POST",

    headers : {
      "Content-Type" : "application/json",
    },

    body : JSON.stringify({
      title: "new product",
      name : "thirumalesh",
      price: 10.11,
      Category : "men's Clothing",
      });
      });

  if(! response.ok) {
    throw new Error(" Failed to post data ");
    }
```

→ tells to the server what type of data you sending

Converts Javascript objects into JSON

// Wait for Response body to be parsed into JSON before Continuing.
Const data = await response.json();

Console.log (data);

}

Catch (error) {

Console.log (error);

}

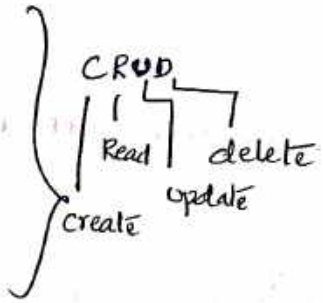
}

→ Converts JSON (strings) from server into Javascript object

→ 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) (or) specific fields
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 proccessed successfully
1) 200 → OK – GET, PUT
2) 201 → Created → POST
3) 204 → No Content → DELETE

3XX → Redirection
Client must takes another action
301 → moved permanantly
302 → Found
304 → Not modified

4XX → Client Errors
Problem from client 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.

1) What is .json() ?

- .json() : Converts JSON (Raw Response) from the server into a Javascript Object
- Converts JSON → Javascript Object

2) 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 formate the body data is in
- Which means, Hey, server, I am sending Json data
- tells the formate of the body being sent
- text/plain → simple text
- application/xml → XML
- application/json → JSON Data

(Request) Content-Type = what iam sending

(Response) Accept = What I am Recieving

2) Recieve :

headers: {

"Accept" : "application/json",

↓

It is tells the format of the body excepected in Respons from server

3) 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) Whats is JSON.stringify() ?

- JSON.stringify() Converts Javascript Object into JSON string

Note:
Fetch cannot sends 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:
- Doesnot need double qoutes for keys.

JSON string
- (Javascript Object Notation) JSON is a text for sending data.
- It look 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 qoutes 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 dont handle errors :

  - API crash
  - ui may stay blank
  - User wont 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-sided Errors
    invalid Request Data

* By using error handling, to prevents the api crashes and improves user Experience.

* try-catch + async/await (Reccommonded).

8) 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:
axio.get(url), axios.post(url, data), axios.put(url, data), axios.delete(u

Syntax:
axios.get(url)
axios.post(url, options)
axios.put(url, options)
axios.patch(url, options)
axios.delete(url, options)

Ex:
1) axios.post("http://api.example.com/users",
{
title: "iphone",
price: 200,
});
2) 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, inceptors, params etc)

Const api = axios.create ({... })

Ex: Const api = axios.create ({

baseURL : "https://fakestoreapi.com",

headers : {

"Content-Type" : "application/json",

"Authorization" : 'Bearer TOKEN"

}

Params : {

Category : "electronics",

Page : 2,

}

});

* To avoiding Repeating Code
* used to predefined Configuration the app.
* Global Configuration.

Axios Features:

1) Automatic JSON handling

Convert Js Object → JSON (request)

Convert JSON → Js Object (Response)

2) Data Transformation

Transform the Request and Responses 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) Inceptors

Modify Requests/Response Globally

7) Less boilerplate code / Cleaner Syntax & Maintaince Easy

FAQ) (JSON.stringify)

In Axios, do we need body, res.json(), headers ?

Ans body → X (NO)
res.json() → X (NO)
} because axios automatically serializes request data and parse JSON responses.

headers → sometimes → need only for authententication 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 (Builin) (no install)	Javascript library (need to install)
2)	JSON.stringify() requeired for JSON request (manually)	2) Automatic
	res.json()	
3)	[redacted] requeired 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 (manully (res.ok)	(7) Automatic
8)	No Request cancel	8) Request Cancel
9)	No timeOut	10) 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: "mens clothing",
   price: 1114.9
  });
 console.log("Product created successfully:", res.data);   // automatically to JSON to JS object

}
catch (error) {
  // Url formate Wrong
  if (error.response)
    {
  console.log("Server error, please try again:", error.response.status);

  } // internet off
  else if (error.request)
     {
     console.log('NO internet Connection', error.request);

     } // JS Code Issues
     else {
      console.log("Some thing went wrong");

     }
  }
}
```

Note: Every Asyn Function always returns promise, even if you dont write a promise().

* Both fetch and axios use GET method by default (if you dont specify the method it will automatically fetch the data cors 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 enhance Component.
* Component → adds extra Behaviour → Return new Component
* Reuse Common Logic

PropTypes: → Show warnings

* PropsTypes is a React Feature used to validate the props passed to a Component.
* PropsTypes are used the excepected type of props in React Components and warn developers if incorrect props are passed.
* Prevent Runtime bugs, improve code Reusability.
* npm install props-types.

Ex: import PropsTypes from "props-types",

```
function student (Props) {

  Return (
      <h2> Props Types </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 PropsTypes:

1) string (Props must be string)
2) number (number)
3) bool (boolen)
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 Props ?

* Default Props are used assign default values to props when no value is passed from the parent Component.
* Prevents undefined errors

Syntax:

```
Student.defaultProps = {
    name: "thiru"
}
```

(OR)

(Modern way):

```
functional Student ({ name = "thiru" }) {
    return ()
}
```

Note:- Parent can Override : if parent passes props, default values are ignored.

What is the difference b/w default Props and Normal props.

Sol

Normal Props:

* Passed data from parent to child
* value is provided explicitly

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	State
1) Pass the data from parent to child	1) Internal data of Component
2) Read-Only (immutable) (Cannot be modified by the child).	2) mutable (can be changed)
3) Controlled by Parent Component	3) Controlled by same Component
4) When Parent Props are updates then Re-render will happen	4) When state changes then re-render will happen

Form handling

Ways to Handle Forms in React.

1) Controlled Form
2) Uncontrolled Form.

1) Controlled Form

* Form Data is controlled by React state.
* input values from usestate
* On everychange → state updates

2) Uncontrolled Form: (DOM controlls the form.

* Form Data is handled by the DOM itself.
* No state for each input
* used useRef
* Used ref to access value
* React doesnot re-renders 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  sendata = {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 updates

3) Unmounting phase
   (Component is Removed)
* When Component disscarpper from screen
* ComponentWillUnMount()
* used for clear timers, Remove event listeners, cleanup

ComponentDidMount() → useEffect([], [])

ComponentDidupdate() → useEffect([], [dependency array]);

Component WillUnmount() → cleanup function

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 ?

* Redux is for large apps , useContext is good for small apps.

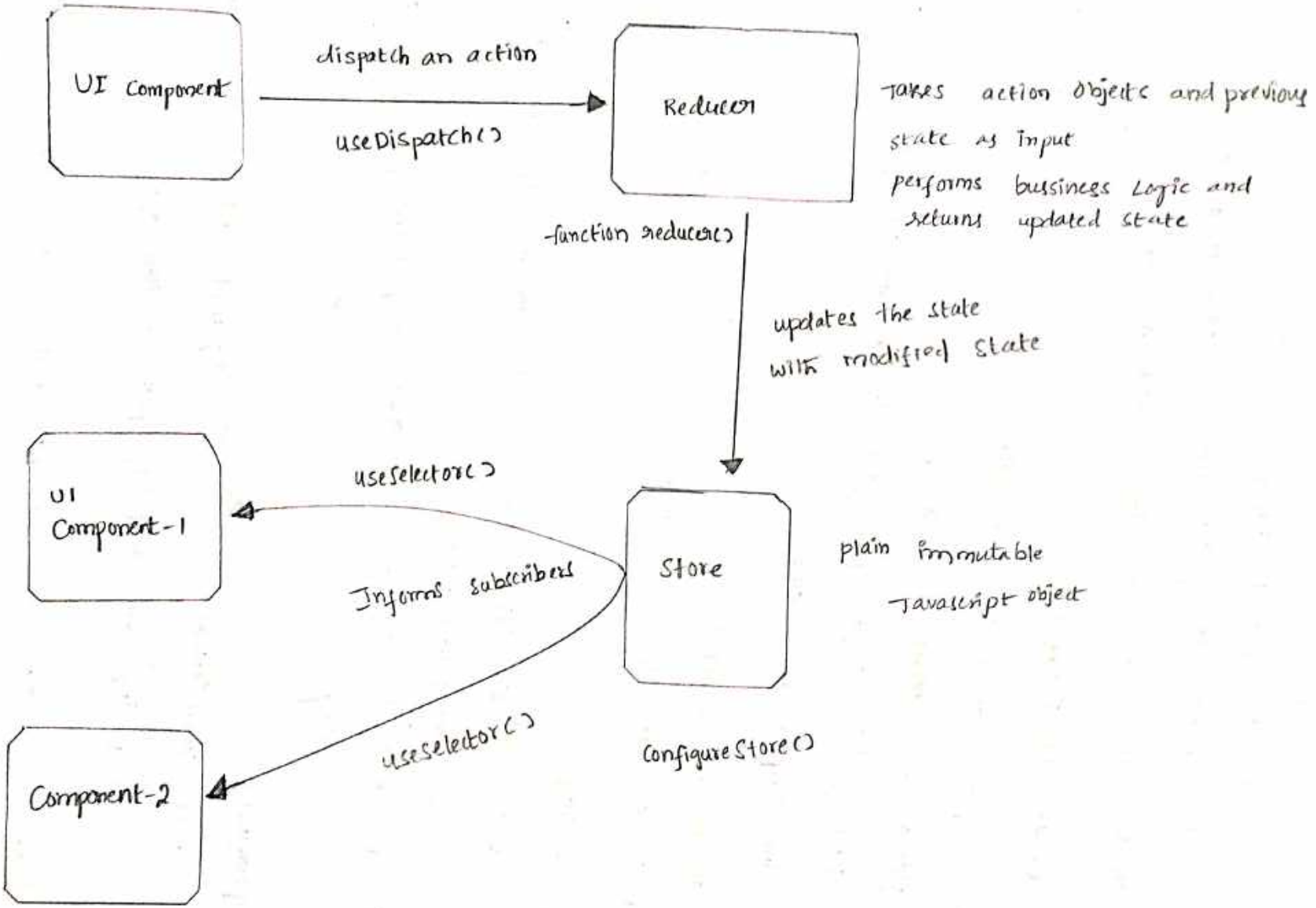
* Props drilling :

  - useContext : If your app is big, you still sometime 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 thy don't need the update → slows app
  - Redux : Only Components using that part of re-render – faster for large apps.

Redux Terminology (store, reducer, actions, dispatch, listeners/ subscribers)



1) Store:

* single source of truth
* Holds or store data for entire application state
* plain immutable javascript object
* Ony 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 sends to action to Redux
* dispatch({ type : "name", payload : "shivamalesh" });

4) Reducer

* reducer is a pure function
* Takes:
  - previous States
  - 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 UI when 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:

```
<h5> Name : {state.name} </h5>
```

Note:

Configure store()

* It is a function provided by Redux toolkit
* used to create Redux store (Replacements of createstore()).
* combined reducers

Core - Redux methods (Not Reccommended) ✓

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() : Listenes 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,
    cart : cartReducer;
});
```

(or)

```
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 recieving 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

* Middle Ware handles all above side Effect

Can Reducers handles async 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 hierachy, while still keeping it in the same Reat tree.

* Portals are ~~avoid~~ used for to avoid CSS and layout Issues

  Overflow: hidden, z-index, position: Relative. etc.

* Props and states still work.

Syntax:

```
ReactDOM.createPortal(
)
```

Ex:

```
import ReactDOM from "react-dom";

function Modal() {
return (
  ReactDOM.createPortal(
    <div.className="modal">
      Hello Modal </div>
  document.getElementById("portal-root")
  );
}
```