APNA COLLEGE SIGMA 6.0
HANDWRITTEN NOTES

DAY 1 – DAY 58

100 Days of Code.

Day 1 08/01/2025, wednesday

HTML – Structure
CSS – Style
JS – Function.

HTML – Hyper Text Markup Language. (Level 1 Part A)

Structure & formatting

HTML Elements

			Tags
Paragraph			
Heading	<h1>		</h1>
Image			

eg This is a paragraph

opening tag — closing tag

content

element.

bold
paragraph   (more than one space (or) next line count work by default)
heading <h1> to <h6> (size of heading)
<h1> </h1>

<html> </html> } Root Tag

HTML Boilerplate :- Standard format or skeleton of coding HTML Code.

```
<!DOCTYPE html>
<html>
    <head>
        <title> My First Page </title>
    </head>
    <body>
        <p> hello world </p>
    </body>
</html>
```

<head> ⇒ to link multiple links. (meta data storage)

Day 2 of 100 09/01/2025 Thursday

HTML (Level 1) Part-B

Lists in HTML

Unordered list (ul)

```
<ul>
    <li> Bread </li>
    <li> Butter </li>
    <li> Jam </li>
</ul>
```

- Bread
- Butter
- Jam

Ordered List (ol)

```
<ol>
    <li> Bread </li>
    <li> Butter </li>
    <li> Jam </li>
</ol>
```

1. Bread
2. Butter
3. Jam

type = "A" (Attribute for A, B, C, D. - - - ordering)
type = "a"
→ inside <ol type = "A">

</ol>

HTML Attribute

Attributes are used to add more information to the tag.

Anchor Element

<a href = "https://google.com"> Google </a>

href ⇒ hyper text reference.

- absolute link.
- relative link.

Image Element

<img src = "image.png" alt = "Random Image">

- source attribute.
- alternative name.

attributes

Height, width

Next line (break line)

Bold
Italic
Underline <u> </u>

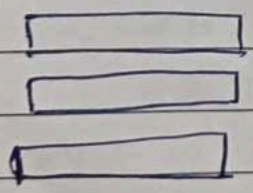
Comments in HTML : This is part of code that should not be parsed.

Ctrl + / to comment a line.

HTML is not case sensitive

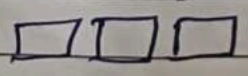
Day 3 of 100 10/01/2025 Friday

Block Elements



* starts from new line

Inline Elements



* Dont start from new line.

Inline	Block
eg:- anchor, Image. Span	eg:- Heading, Paragraph, div

Div element (content division element)
Div is a container used to hold other HTML elements or groups together. It is a block element.

Span Element :
Generic container used to hold other HTML elements or group elements together.
It is an Inline Element

HR Tag (Horizontal Rule)
<hr>
Useful for page breaks.

Subscript
H_2O ⇒ H ₂ O

Superscript
a^2 ⇒ a ²

Semantic Markup :- It is the markup that relates to the meaning of content.

SEO friendly ⇒ Search Engine optimization.

Semantic Tag

<header> </header> <footer> </footer>
<main> </main> <nav> </nav>

<article> </article>
<section> </section>
<aside> </aside>

HTML Entities

  ⇒ space (non breaking space)
< ⇒ lesser than
> ⇒ greater than
" ⇒ " quote

Day 4 of 100 11/01/2025 (Saturday)

Emmets

Child Emmet Abbreviation (>)

ul>li ⇒ <ul>
<li> <li>
</ul>

Sibling Emmet Abbreviation (+)

h1+p ⇒ <h1> </h1>

Multiplication Emmet Abbreviation.

h1*5 ⇒ <h1> </h1>
<h1> </h1>
<h1> </h1>
<h1> </h1>
<h1> </h1>

HTML5 ⇒ DOCTYPE HTML

HTML Standard is a document that tells to the browser how HTML should work.

HTML Level 3

Tables : Are used to represent real life table data.

column ↓

→ row

```
<table>
  <caption> Table Caption </caption>
  <tr>
      <th> header 1 </th>
      <th> header 2 </th>
  </tr>

  <tr>
      <td> data 1 </td>
      <td> data 2 </td>
  </tr>
</table>
```

<tr> ⇒ used to display table row
<td> ⇒ used to display table data
<th> ⇒ used to display table header.

Semantics in Table

<thead> ⇒ to wrap table header
<tbody> ⇒ table body
<tfoot> ⇒ table foot.

Colspan & Rowspan Attributes

rowspan = " "
colspan = " "

Item ⇒ rowspan = 2
Price ⇒ colspan = 2

Food Menu

Item	Price	
	INR	USD
A	1	0.1
B	2	0.2
C	3	0.3

Forms in HTML

<form> <|form>

action attribute :- what to do after submission

Form elements

1) Input elements :- used to create multiple form controls.

<input>
<input type = " " >

Types of input Element

* text
* password
* number
* time
* color
* Button

2) Placeholder attribute.
placeholder = " " (attribute)

3) Labels :- If we click on label name, it will activate form

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

Classical Approach

```
<label for = "____" >          </label>
                 ↓
                 id
```

4) Button element

```
<button>          </button>
```

types

* submit ⇒ to submit
* button ⇒ to do something (to create normal button inside a form)
* reset ⇒ to do something. (to reset form)

Input type as button

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

to submit

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

Day 05 of 100 12/01/2025 (Sunday)

Name Attribute :- Name of form control. Submitted with the form as part of name/value pair

<input type = "name" placeholder = " enter name"
id = "username" name = "username" >

name attribute comes in url.

Checkbox Attribute ☑ ⇒ will sent on ☐ ⇒ won't sent off.

<input type = "checkbox" name = "age" id = "age"
checked >
<label for = "age" > I am 18+ </label>

by default checked. (not compulsory)

Radio Attribute ◉

<input type = "radio" name = "fruit" id = "apple"
value = "apple" >
<label for = "apple" > Apple </label>

name = " " should be same inorder to group all radio elements (for selecting one option.)

value attribute returns the value selected by using Radio button.

Select Element (Dropdown)

```
<select name = "profession" id = "profession">
<option value = "student" > Student </option>
<option value = "dev" > Developer </option>
</select>
```

Range Element

```
<label for = "volume" > Volume </label>
<input type = "range" min = "0" max = "100" name = "vol" >
```

step = "10" to increase by 10

value = "20" to keep default value as 20.

Day 06 of 100 13/01/2025 (Monday)

CSS – Getting started with CSS

Cascading Style Sheets.

Basic CSS Format

```
h1 {
    color : red;
}
```

selector

color → property

red → value.

How to include Styles?

* Inline Styles
  <h1 style = " color : red " > College </h1>

* using <style> tag
  Style is added using the <style> element in the same document.

```
<style>
    h1 {
        color : red;
    }
</style>
```

* External Stylesheet
  - writing CSS in a separate document & linking it with HTML file.

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

relation

Color in CSS

```
h1 {
color : blue ;
}
```

Background color

```
h1 {
background - color : yellow ;
}
```

CSS color system

Red + Green = Yellow
Red + Blue = purple
Blue + Green = light blue
Red + Blue + Green = white. (255, 255, 255)
Black = (0, 0, 0)

Teal
olive
maroon.

Hexacode
color = #FFFFFF

letters: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Text Properties

text align
font-weight
text-decoration
line-height
letter-spacing
font-size.

text-align: left/start
: right / end
: center
: justify.

} relative to parent element

Day 07 of 100 14/01/2025 (Tuesday)

Font weight (100 - 900)

font-weight : normal //400
: bold //700
: bolder } parent
: lighter }

Text-decoration

text-decoration : underline — hello
text-decoration : none — hello
text-decoration : overline — hello
text-decoration : line-through — ~~hello~~

double
dashed
dotted
wavy

Line height

controls the height of the line of text.

line-height : normal
line-height : 2.5 ⇒ 2.5 x normal

Letter spacing

controls the horizontal spacing behaviour between text characters.

letter-spacing : normal
letter-spacing : 10px.

Font-size units in CSS

Absolute	Relative
px	%
pt	em
	rem

px - pixel (globally fixed)

1 inch = 96 px
1 inch = 72 pt

em
rem ⇒ parent dependent

px ⇒ not for responsive website.
↳ normal font size ⇒ 15 px

font-family (Arial, Times New Roman ---)
Specifies a prioritized list of one or more font family name.

cssfontstack.com

Day 08 of 100 15/01/2025 (Wednesday)

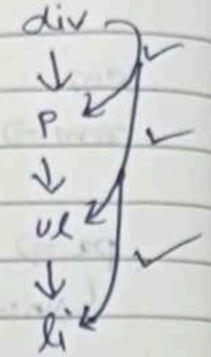
Selectors in CSS

* h1 {}	* {}	id="sk" #sk {}
3 element selector	3 Universal Selector	3 Id selector
class="sk" .sk {}		
3 class selector		

→ multiple level

Descendant Selector (direct and indirect child)

eg:- Selects all paragraphs inside divs



outer level inner level

eg div p { } nav ul li { }

eg .class a { }

Sibling Combinator (Adjacent)

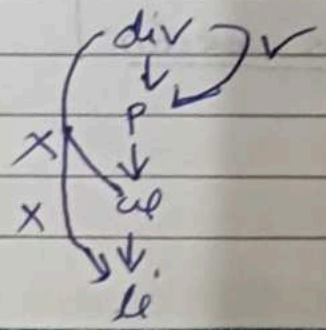
p + h3 { }

Heading 3 that comes immediately after any paragraph

  <h3>
  </h3>
} Same level. (Sibling Relation)

Child Combinator (direct descendants)

span > button { }



Attribute Selector

Select elements based on the presence or value of a given attribute.

square bracket

input [attr = "value"] {

}

Day 01 of 100 16/01/2025 (Thursday)

Pseudo class

A keyword added to a selector that specifies a special state of the selected elements.

:hover
:active
:checked
:nth-of type.

element : pseudoclass {

}

active ⇒ click

checked ⇒ for radio, checkbox button.

HTML

```
<input type = "radio" id = "yes" name = "agree" >
<label for="yes" > Yes </label
<input type = "radio" id = "no" name = "agree" >
<label for = "no" > No </label >
```

CSS

```
input [type = "radio"] : checked + label {
    color : red;
    font-weight : bold;
}
```

nth-of-type pseudo class

- Class : nth - of - type (n) {

  }

Pseudo Element

A keyword added to a selector that lets you style a specific part of the selected elements.

:: first letter.
:: first line.
:: selection. => changes CSS for selected part.

Cascading

The CSS cascade algorithm's job is to select CSS declarations in order to determine the correct values for CSS properties.

```
h2 {
    colour : yellow;
}
```

h2 {
color : orange ; ⇒ finally orange
}

Selector Specificity

Specificity is an algorithm that calculates the weight that is applied to a given CSS declaration.

inline style <

① id > class > element.

② If same specificity, then cascading

id	class, attribute & pseudo class	element & pseudo element.

!important

To show the most specific thing in the document.

h2 {
background - color : blue ! important ;
}

cannot be changed.

!important > inline style > id > class > element.

Inheritance in CSS

```
<div>
    <p>
        <ul>        </ul>
    </p>
</div>
```

 ←
↓
<p>
↓
<ul>

exceptions
* button
* input.
to inherit from parent

```
input {
    background-color : inherit ;
}
```

Exception properties
* width
* height
* border.

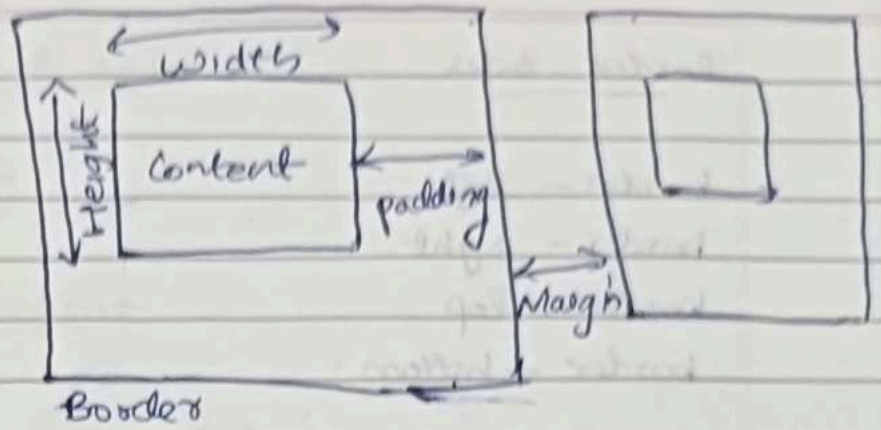
Day 10 of 100 17/01/2025 (~~Thursday~~ Friday)

CSS Part 3

Box model in CSS.

*Height

* width
* Border
* Padding
* Margin



Margin :- brown Padding :- green.
Content : Blue.

Height & width

```
div {
    height : 100px;
}
div {
    width : 500px;
}
```

Border
used to set an elements border.
* border - width
* border - style
* border - color

```
div {
    border - width : 10px;
    border - style : solid;
    border - color : red;
}
```

Shorthand

```
~~border~~
div {
    border : 10px solid red;
}
```

Border Sides

border - left
border - right
border - top
border - bottom.

Border Radius

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

used to round the corners of an element's outer border edge.

Padding Property

- padding - left
- padding - right
- padding - top
- padding - bottom.

```
div {
    padding - left : 50 px;
}
```

Padding Shorthand

1) for all 4 sides
padding : 50 px;

2) top & bottom | left & right

padding : 1px 2px;

3) top | left & right | bottom

padding : 1px 2px 3px ;

4) top | right | bottom | left (clockwise)

padding : 1px 2px 3px 4px;

Margin Property

The set spacing on the outside of border.

- margin - left
- margin - right
- margin - top.
- margin - bottom

```
div {
    margin - left : 50px;
}
```

Day 11 of 100 18/01/2025 (Saturday)

Inline v/s Block

Inline - Takes up only neccessary width

Block line — Takes up full-width available (whole block)

Display
It sets whether an element is treated as a block or inline element and the layout used for its children.

display : inline
display : block

For inline element

Margin only applies on Right and left and it wont apply on top & bottom

~~Margin~~ padding applies on left and right, but on top & bottom it overlaps.

span

For span
height & width cannot be adjusted.

If div is inline ⇒ margin wont work for top & bottom

display
inline-block
element style block
behave inline
single line.

display
none

~~Relative units~~

Units in CSS

Absolute	Relative
px	%
	em
	rem

Percentage (%)
It is often used to define a size as relative to an elements parent object.

em (text related properties.)
Font size of parent.

drawback of em
snowball effect in nested element.

Rem (Root Em) ⇒ Root element.

Display inline for list elements

→ Li (List items) have their display property set to "list-item" by default which makes them a bulleted list item. When the property is reset to "inline", then the bullet points also disappear. In a way, we override the default settings with CSS.

Margin : 0 auto; Center of the page.

Day 12 of 100 CSS – Part 4 Sunday 19/01/2025

CSS – Next steps

Alpha channel
Sets opacity for a color.

Alpha ⇒ 0 to 1
rgba (255, 255, 255, 0.3)

00 to ff

Opacity
Sets opacity for any element.

Range 0 to 1

Transitions in CSS:
Transition enables you to define the transition between two states of the element.

Transition:
(1) state → (2) state

transition-duration :

Transition: property name | duration | timing-function | delay

CSS Transform

This property lets you rotate, scale, skew or translate an element.

(i) Rotate

transform : rotate (45 deg)

+ degree

- degree

(ii) Scale

transform : scale (0.5);

1 ⇒ normal size

(iii) Translate to move an element to x or Y axis.

transform : translate (50px, 50px)

(iv) ~~Transform :~~ Skew

transform : skew (30 deg) ;

Box shadow

It adds shadow effects around an element's frame.

box-shadow : 2px 2px 10px green.

- 2px → x offset
- 2px → Y offset
- 10px → Blue radius.

Background image

background-image : url ( );

background-size : contain / cover / auto.

Download images :- unsplash

contain :- To fit all image.
cover :- to cut or crop.
auto :- to stretch.

Day 13 of 100 Monday 20/01/2025

CSS Part 4

Position
The position CSS property sets how an element is positioned in a document.
The top, right, bottom and left properties determine the final location of positioned element.

- static
- relative
- absolute
- fixed.

Static
The top, left, bottom, right and z-index properties have no effect.
This is the default value.

Relative
The offset is relative to itself based on the values of top, right, bottom and left.

Absolute
The element is removed from the normal document flow, and no space is created for element in the

page layout.
It is positioned relative to its closest positioned ancestor. if any; otherwise, it is placed relative to the initial containing block.

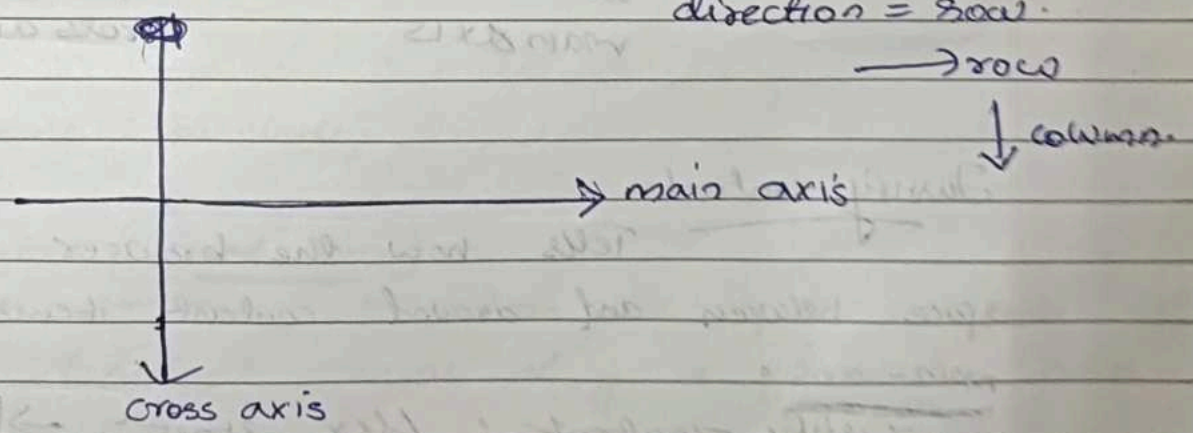
Fixed
The element is removed from the normal document flow, and no space is created for the element in the page layout.
It is positioned relative to the initial containing block established by the viewport.

Day 14 of 100 Tuesday 21/01/2025

Flexbox in CSS. (to make responsive website)

Flexbox layout :- It is a one dimensional layout method for arranging items in rows or columns.

Flex model



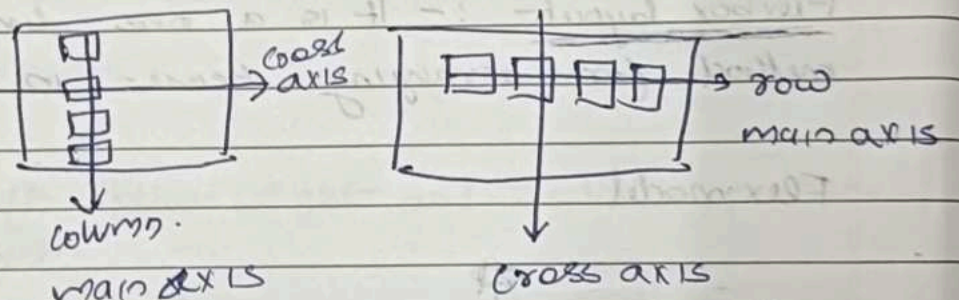
Display Flex (direct child)

display : flex ;

Flexbox direction

It sets how flex items are placed in the flex container, along which axis and direction.

- flex-direction : row ; (main axis = left to right)
  [abcd]
- flex-direction : column ; (cross axis = top to bottom)
  [a b c d]
- flex-direction : row-reverse ; (main axis = right to left)
  [dcba]
- flex-direction : column-reverse (cross axis = bottom to top)
  [d c b a]



Justify content

Tells how the browser distributes space between and around content items along the main-axis.

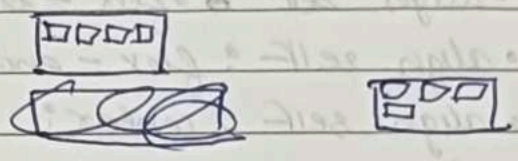
- Justify-content : flex-start ; → [abcd]
- Justify-content : flex-end ; → [abcd]
- Justify-content : center ; → [abcd]

- Justify-content : space-between;
- Justify-content : space-around;
- Justify-content : space-evenly;

Flex wrap

Sets whether flex items are forced onto one line or can wrap onto multiple lines.

- flex-wrap : nowrap;
- flex-wrap : wrap;
- flex-wrap : wrap-reverse;



↳ cross axis ⇒ reverse direction.

Day 15 of 100 wednesday 22/01/2025

Align Items

Distribute items along the cross axis.

align-items: flex start;
align-items : flex-end;
align-items : center;

align-items : baseline;

Align content (cross axis)

It sets the distribution of space between and around content items along a flexbox's cross-axis.

- align-content : flex-start / flex-end / center.

- align-content : space-between / space-around / evenly.
- align-content : baseline;

Align Self (Individual property) (high priority than align items)
Aligns an item along the cross axis.

- align self : flex-start ;
- align self : flex-end ;
- align self : center;
- align self : baseline ;

Flex sizing

- Flex Basis (main axis)
  It sets the initial main size of a flex item.

  flex-basis : 100px ;

- Flex grow
  It specifies how much of the flex container's remaining space should be assigned to the flex item's main size.

  flex-grow : 1;

- Flex-shrink
  It sets the flex shrink factor of a flex item.
  flex-shrink : 1;

Flex Shorthand

- flex-grow | flex-shrink | flex basis
  flex : 2 2 100px;

- flex grow | flex basis
  flex : 2 100px;

- flex-grow (unitless)
  flex : 2;

- flex-basis
  flex : 100px;

Day 16 of 100 23/01/2025 (Thursday)

CSS Part 6

CSS Grid (2D)

Setting a container's display to grid will make all children grid items.

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

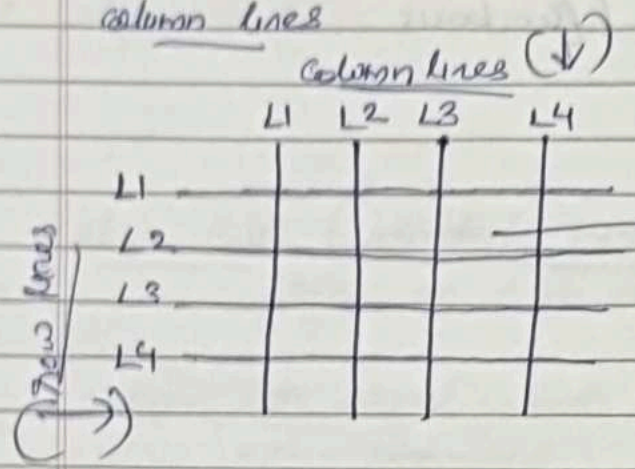
By default (vertical)

No. of lines = No. of tracks + 1

Grid Model

Grid lines
Grid cells
Grid Track

column lines



grid track → distance between two tracks

Grid Template

They define the lines & track sizing.

- grid-template-rows : 100px 100px 100px;
- grid-template-columns : 50px 50px 50px;

track size

Repeat (used to divide all available space).

grid-template-rows : repeat (count, 1fr)
grid-template-columns : repeat (count, 1fr)

grid-template-rows - repeat (2, 1fr) → 1 fraction of 100% width.
grid-template-rows : 1fr 1fr

both are same.

Grid Gaps (They define gaps between the lines)

row-gap : 10px;
column-gap
grid-gap : rowGap columnGap;

Day 17 of 100 24/01/2025 (Friday)

Grid Columns

Defines an item's starting & ending position inside the column

grid-column-start : line-number ;
grid-column-end : line-number ;

grid column : 1/3 ;
(1 = start, 3 = end)

grid column : 1/ span 3
(1 = start, 3 = no. of columns)

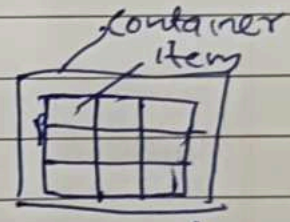
Grid Rows

Defines an item's starting & ending position inside the row

grid-row-start : line-number ;
grid-row-end : line-number ;

grid row : 1/3
grid row : 1/ span 3

Common Properties



Grid

~~Container~~

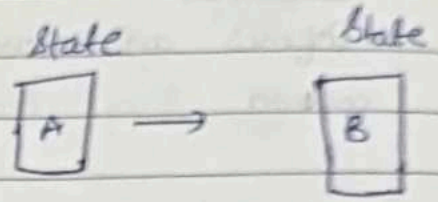
- Justify-items (container) } horizontal
- Justify-self (item)

- align-items (container) } Vertical
- align-self (item)

- place-items } for both Horizontal and Vertical.
- place-self

Animations in CSS

To animate CSS elements.




```
@ keyframes myName {
    from { font-size : 20px ; }
    to { font-size : 40px ; }
}
```

- animation-name
- animation-duration
- animation-timing-function.
- animation-delay.
- animation-iteration-count
- animation-direction.

Animation shorthand

animation : myName 2s linear 3s infinite normal

name, duration, timing function, delay, iteration count, direction

Percentage in Animations

```
@ keyframes myAnimation {
    0% {    ;}
    50% {   ;}
    100% {  ;}
```

~~Day 18~~

Media Queries

Help create a responsive website.

media feature - width (of viewport)

```
@media (width : 400px) {
    div {
        background-color : red;
    }
}
```

orientation
- vertical — portrait
- horizontal — landscape

(min-width) (max-width)

orientation (of viewport)

```
@media (orientation : landscape) {
    div {
        background-color : red;
    }
}
```

z-index

It decides the stack level of elements.
Overlapping elements with a larger z-index cover those with a smaller one.

z-index : auto (0)
z-index : 1/2/ - - -
z-index : -1/-2/ - - -

Day 18 of 100 25/01/2025 (Saturday)

Basic Design Principles

1) Color
2) Typography

Color Theory
color wheel.

Red ⇒ excitement, danger
black, grey, white ⇒ rich ~~luxury~~ luxury

** Monochromatic, Complementary, Triadic

Day 19 of 100 26/01/2025 (Sunday)

Typography google-fonts

font-family : sans-serif ;

link embed

Icons
google-icons, font awesome. cdn JS

To create menu (without JS)

```
<div>
    <input type="checkbox" id="check">
<div>
  <label for="check" style="color: red">
    <i class= ———— ></i>
  </label> (icon)
</div>
```

```
<div>
  <label for="check" style="color: yellow">
    <i class= ———— ></i>
  </label>
</div>
```

Day 20 of 100 27/01/2025 (Monday)

Bootstrap ⇒ framework ⇒ frontend tool kit.

* button
* navbar
* cards
* forms

} prebuilt components.

getbootstrap.com

— works on flexbox concepts.

Benefits
* Faster & Easier
* Pre-build components.
* browser compatability.

button ⇒ disabled

cdn ⇒ content delivery network

button ⇒ btn

Components	Layouts
pre-built and reusable piece of user-interface (UI)	overall structure and arrangement of the elements.
eg:- button, forms.	eg:- containers, grid.

Container

Layout

Containers are used to contain, pad, and (sometimes) center the content within them.

Hello

Button class="btn"

Danger ⇒ Red color	warning ⇒ yellow orange.
Success ⇒ green	
Secondary ⇒ Gray	
Primary ⇒ Blue.	
Info ⇒ light blue	
light ⇒ white	
Dark ⇒ black	

Badges (for notifications etc).

```
<h3> Hello <span class = "badge text-bg-danger" > 15+
</span> </h4>
```

Hello [15+] → red

Day 21 of 100 28/01/2025 (Tuesday)

- Alerts

```
<div class = "alert alert-danger" role = "alert">
    Failed!
</div
```

- Button Group

```
<div class = "btn-group" >
  <button class = "btn btn-success" > one </button>
  <button class = "btn btn-success" > two </button>
  <button class = "btn btn-success" > three </button>
</div>
```

one	two	three

outlined-button-group

mb-3 ⇒ margin bottom - 3

- Navbar
- Card

- Grid
  12 column system.
  grid options

- col-
- col-sm-
- col-md-
- col-lg-
- col-xl-
- col-xxl-

col-sm-3 col-md-4 col-lg-6

Grid (Vertical Alignment) (Columns)

Select
to create drop down and select an option from the list.

Checkbox and radio

layout

Day 22 of 100 29/01/2025 (Wednesday)

CSS Project (Spotify clone)

Overflow : hidden ; ⇒ for height restriction and stop scrolling

flex : 1 ; ⇒ for giving complete remaining space by giving only content space for others.

1rem = 16px
root

overflow : auto ; ⇒ for scrolling if needed.

Day 23 of 100 30/01/2025 (Thursday)

Sticky Navbar

Cards

Footer - Line

Range (Input)
⇒ Appearance property

appearance : none ;

Slider

quirksmode.org
styling_and_scr.html.

slider

Day 24 of 100 31/01/2025 (Friday)

Javascript

Using the Console

Uses REPL
Read - Evaluate - Print - Loop

clear screen => ctrl + L

What is variable?
* A variable is simply the name of a storage location.

Datatypes in JS

Primitive types

- Number
- Boolean
- String
- Undefined
- Null
- Bigint
- Symbol

typeof ⇒ to check data type of variable.

Numbers

- +ve ⇒ 14, 15
- -ve ⇒ -14, -12
- floating numbers ⇒ 4.6, -8.9

Day 25 of 100 1/02/2025 (Saturday)

Operations in JS

addition
sum = a+b

subtraction
diff = a-b

multiplication
prod = a*b

division
div = a/b

modulus
rem = a%b

Exponentiation

pow = a**b

NaN in JS

The NaN global property is a value representing Not-A-Number.

0/0.

NaN-1 NaN + NaN

Nan*1

Operator Precedence

General order of solving an expression.

( )
**
*, /, %
+, -

eg:-
(2+1) * 3 = 9
3/1 + 2**2 = 7
4+1 *6/2 = 7

let Keyword

Syntax of declaring variables.

```
let age = 23;
age = age + 1;

let sgpa;
sgpa = 9.39;

let num1 = 1;
let num2 = 2;
let sum = num1 + num2;
```

const Keyword

values of constants can't be changed with re-assignment & they can't be redeclared.

```
const year = 2025;  →  year = 2026; //error
const pi = 3.14;       year = year + 1; //error
const g = 9.8;
```

var keyword

Old syntax of writing variables.

var age = 25;

var cgpa = 9.39;

var num1 = 1;

var sum = num1 + num2;

Assignment Operators

age = age + 1 age += 1	age = age - 1 age -= 1	age = age % 3 age %= 3
age = age * 1 age *= 1	age = age / 2 age /= 2	

Unary operator (one operand)

Pre age = age + 1 Post age ++	age = age - 1 age --	binary operator (two operand)
Pre ++age	--age	

Post ⇒ (use, then change)

Pre ⇒ (change, use)

Identifier Rules

All JavaScript variable must be identified with unique names (identifiers)

- Name can contain letters, digits, underscores, and $ (no space)
- Must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables.)
- Reserved words (like JS Keywords) cannot be used as names.

age ✓ age123 ✓
age_ ✓ 12age ✗
_age ✓ age is ✗
$age ✓

way of writing identifiers: camelCase, snake_case, PascalCase

Boolean in JS (true (or) false)

```
let isAdult = true;
isAdult
true
```

* Type change is possible in JS.

Typescript
Static Typed, where JS is dynamic typed.
* designed by Microsoft

Static typed variables cannot change type.

dynamic typed variables can change type

String in JS
Strings are text or sequence of characters.

```
let name = " Tony Stark";
let char = 'a';

let num = " 2359 ";
```

" " string
' '

```
let newString = " Shane 'Sunny' Kottuppallil ";
```

String indices (0 based indexing)

```
let name = " Shane Sunny "
```

S	h	a	n	e	_	S	u	n	n	y
0	1	2	3	4	5	6	7	8	9	10

name [0] = "s" , name [1] = "h"

no. of indices = length + 1

length

name . length

concatenation

"Shane" + " " + "Sunny";

null & undefined in JS

undefined

A variable that has not been assigned a value is of type undefined.

null

The null value represents the intentional absence of any object value

To be explicitly assigned.

Day 26 of 100 02/02/2025 (Sunday)

console . log()

To write (log) a message on the console.

```
cosole . log(" Shane Sunny");
console . log(" 2+2 ");
console . log( " Shane ", " Sunny " , 4);
```

Linking JS File

<script src = "app.JS" > </script>

Template literals

They are used to add embedded expressions in a string.

```
let a=5;
let b = 10;
// console.log ( "Price is", a+b , "rupees ");
console.log ( `Your pay ${a+b} rupees`);
```

backtick backtick

'${ }' → Template literal. backtick is used

More Operators in JS

Comparison

> Greater than
>= Greater than or equal to
< lesser than
<= lesser than or equal to
== equal to (wont compare data type)
!= not equal to

ASCII { A - Z a - Z
97 122 65 90

Exceptions

(5 == '5') ⇒ This is true. The value is same, the data type is disregarded.

If you want to make strict comparisons with value and data type then use ===

(5 === '5') ⇒ This is not true. This is false.

(0 == ' ') ⇒ This is true.

(0 == false) ⇒ This is true.

(null == undefined) ⇒ This is true.

Comparison for non-numbers

'a' > 'A'	'B' < 'C'	a ⇒ 65
true	true	Z ⇒ 90
'a' > 'b'	'*' < '&'	A ⇒ 97
false	false	Z ⇒ 122
'b' < 'c'		
true		

ASC'II value.

JavaScript Unicode

"a" ⇒ 61
"A" ⇒ 41

Day 27 of 100 03/02/2025 (Monday)

Conditional Statements

- if-else
- nested if-else
- switch

if statement

```
if (condition) {
    //Do something
}
```

else if statement

```
if (condition 1) {
//Do something
}
else if (condition 2) {
// Do something else
}
else if (condition 3 {
// Do something else
}
```

else

```
if (condition {
    //Do something
}

else {
    //Do something else
}
```

Logical operator

Logical && (AND)

A	B	A&&B
true	true	true
true	false	false
false	true	false
false	false	false

```
if (marks >= 80 && marks < 90)
{
  console.log("Grade : A+");
}
```

Logical || (OR)

A	B	A||B
true	true	true
false	true	true
true	false	true
false	false	false

```
if (num % 2 == 0 || num % 4 == 0)
{
  console.log("Number is even!");
}
```

Logical ! (NOT)

A	!A
true	false
false	true.

!(false)	!(true)
true	false.

---

truthy & falsy

Everything in JS is true or false (in boolean context). This doesn't mean their value itself is false or true, but they are treated as false or true if taken in boolean context.

Falsy values
false, 0, -0, 0n (BigInt value), "" (empty string), null, undefined, NaN

Truthy values
Everything else

Switch Statements

Used when we have some fixed values that we need to compare to.

```
let color = "red";
switch (color) {
    case "red":
        console.log ("stop");
        break;
    case "yellow":
        console.log ("slow down");
        break;
    case "green":
        console.log ("go");
        break;
    default:
        console.log ("Broken Traffic Light");
}
```

Alerts & Prompt

Alert display an alert message on the page.
alert ("something went wrong!");
Prompt displays a dialog box that asks user for some input.
prompt ("please enter your roll no.");

console.log ("Hello");
console.error "
console.warn "

prompt ("Enter your name:");
alert ("This is an alert");

Day 28 of 100 04/02/2025 (Tuesday)

String Methods

Methods :- actions that can be performed on objects.

Format
stringName.method()

let msg = " Hello"
1) str.trim() ⇒ start and end
trims whitespaces from both ends of string & returns a new one.

let msg = " Hello "
msg.trim()
'Hello'
msg
' Hello '

output : "Hello", but value of msg remains the same.

In JavaScript Strings are immutable.

Strings are immutable in JS.
- No changes can be made to strings.
Whenever we do try to make a change, a new string is created and old one remains same.

2) str.toUpperCase () "RANDOM"
str.toLowerCase () "random"

String Methods with Arguments

Argument is a some value that we pass to the method.

Format
stringName.method (arg)

3) indexOf () method
Returns the first index of occurence of some value in string. or gives -1 if not found.

```
let str = "Ilove coding";
str.indexOf("love")    //1
str.indexOf("J")       //-1 (not found)
str.indexOf("o")       //2 (only 1 index)
```

Method Chaining
Using one method after another. Order of execution will be left to right.

str.toUpperCase().trim()

(Starting index)
(or)
(Starting index, ending) (-1) index based

4) Slice Methods (.slice())

Returns a part of the original string as a new string.

let str = "Ilove Coding";

str.slice(5) // Coding (Starting index = 5)

str.slice(1,4) // love (starting index = 1, (ending index - 1) = 4)

str.slice(-num) = str.slice(length - num).

- Replace & Repeat Method

5) replace (only for first occurence)
Searches a value in the string & returns a new string with the value replaced.

let str = "Ilovelcoding";

str.replace("love","do"); // "Idocoding"
str.replace("o","x"); // "Ilxvelcoding"

6) repeat() method
Returns a string with the number of copies of a string.

let str = "Mango";

str.repeat(3) // "MangoMango Mango"

Day 29 of 100 (Wednesday) 05/02/2025

Arrays (Data structure)
Linear collection of items

let students = ["shane", "sven", "sonny", "shalets"];

shane	sven	shalets	sonny
0	1	2	3

students[1] => sven

typeof(students) => object

students.length => 4

Mixed array
let mixedArray = ["shane", 1, 1.934];

for JS array can take different multiple data types.

* Arrays are mutable unlike strings

Array Methods

1) Push : add to end
2) Unshift : add to start
3) pop : delete from end & returns it.
4) shift : delete from start & returns it.

arr.push ("share");

5) indexOf : returns index of something

primary.indexOf("yellow"); ⇒ return index of element yellow.

↓
array name

returns -1 if value doesn't exist.

6) includes : search for a value

primary.includes("red");
⇒ true
primary.includes("elephant");
⇒ false

reverse function ⇒ changes the original array.
splice ⇒ "
sort ⇒ "

7) concat : merge 2 arrays. (new array is created)

primary.concat(secondary);
primary → first array
secondary → second array

8) reverse : reverse an array. (Changes original array)
primary.reverse();

⇒ reverses the 'primary' array.

9) Slice in Arrays

let colors = ["red", "yellow", "blue", "orange", "pink", "white"];

colors.slice();
▷6 ['red', 'yellow', 'blue', 'orange', 'pink', 'white']

colors.slice(2);
['blue', 'orange', 'pink', 'white'].

colors.slice(2,3);
['blue']

color.slice(-2);
['pink', 'white']

10) Splice in Arrays (gets update Array)

Splice : removes / replaces / add elements in place.

Splice (start, delete count, item 0 --- item N)

eg1 colors.splice (4);
['pink', 'white'];

eg2 colors.splice (1,1);
['red', 'blue', 'orange', 'pink', 'white']

eg3 colors.splice (0,0, "black", "brown", "maroon");

['red', 'yellow', 'blue', 'orange', 'pink', 'white', 'black', 'brown', 'maroon']

11) Sort in Arrays (changes the array)

sort : sorts an Array

let days = ["monday", "sunday", "wednesday", "tuesday"];

days.sort();
⇒ ['monday', 'sunday', 'tuesday', 'wednesday']

let squares = [25, 16, 4, 49, 36, 9]

squares.sort();
⇒ [16, 25, 36, 4, 49, 9]

Array References (address in memory)

[] === []
⇒ false
[] == []
⇒ false.

```
let arr = ['a', 'b'];
let arrcopy = arr;

console.log(arr, arrcopy);
⇒ ['a','b'] ['a','b']

arrcopy.push('c');
console.log(arr, arrcopy);
⇒ ['a','b','c'] ['a','b','c']
```

Constant Arrays

```
const arr = [1,2,3,4,5];

arr = [5,6,7];  ⇒ This throws error
```

but

```
arr.push()
arr.pop()
```
} all these wont throw error.

Nested Arrays (Multi dimensional array)
arrays of arrays.

```
let nums = [ [2,4] , [3,6] ,[4,8]] ;
```

	0,0	0,1
	2	4
1,0	3	6
2,0	4	8

1,1

2,1

Day 30 of 100 (Thursday) 06/02/2025

JS Part 4

For loops
used to iterate a piece of code.

```
for ( let i=1 ; i<=5 ; i++ ) {
    console.log (i) ;
}
```

Infinite loops

```
for ( let i=1 ; i>=0 ; i++) {

}
```

Nested For loops

```
for(let i=1; i<=3; i++) {
    for( let j=1 ; j<=3 ; j++) {
        console.log(j);
    }
}
```

Day 31 of 100 (Friday) 07/02/2025

While loop

```
while (Condition) {
  //do something
}

let i=1;
while (i<=5) {
  console.log(i);
  i++;
}
```

Favorite movies game (guess the movie)

```
const favmovie ="avatar";
let guess = prompt ("guess my favorite movie");
while (guess != favmovie) && (guess != "quit")) {
  guess = prompt ("wrong guess. Try again");
}
  if (guess == favmovie) {
```

```
    console.log("congrats");
} else {
    console.log("you quit");
}
```

Break Keyword

```
let i=1;
while (i<=5) {
    if(i==3) {
        break;
    }
    console.log(i);
    i++;
}
```

Loops with Arrays

```
let fruits = ["mango","apple","grape","pineapple",
              "banana"];

for(let i=0; i<fruits.length; i++) {
    console.log(fruits[i]);
}
```

~~Next~~

Nested loops with Nested arrays:

```
let marks = [ [1,2,5,3], [10, 12, 36, 53],
              [100, 123, 240, 356] ];

for(let i=0; i< marks.length ; i++) {
  for(let J=0; J< marks[i].length ; J++) {
     console.log (marks[i][J]);
  }
     console.log ("\n");
}
```

For of loop

```
for(element of collection) {
    //do something
}
```

```
let fruits = [ "Mango", "apple", "grape", "banana",
               "pineapple" ];

for(fruit of fruits) {
    console.log(fruit);
}
```

Nested for-of loop

```
for ( i of marks) {
    for (J of i ) {
    console.log ( i );
  }
```

Todo list (only JS)

- list – to show all todos
- add – to add a todo
- delete – to delete a task
- quit – to exit the todo.

Day 32 of 100 (Saturday) 08/02/2025

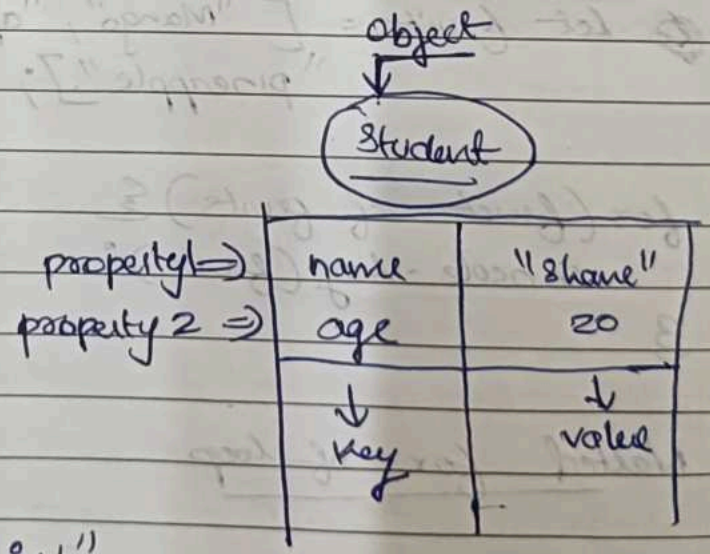
Object Literals

Used to store keyed collections & complex entities.

property ⇒ (key, value) pair

Objects are a collection of properties.

eg let arr = [];
typeof arr
⇒ 'object'



JS Object literals

```
let delhi = {
    latitude : "28.7041° N",
    longitude : "71.1025° E".
};
```

```
const student = {
    name: "shane",
    age : 20,
    marks : 97.7
};
```

Create a Post (Task)

Create an object literal for the properties of thread / twitter post which includes –

- username
- content
- likes
- reposts
- tags.

Get Values

```
let student = {
    name : "shane",
    marks : 99.9,
};

console.log (student.name);
or
consolelog (student["name"]);
```

Conversion to Get Values

JS automatically converts object keys to strings. Even if we made the number as a key, the number will be converted to string.

Add / Update Value

```
const student = {
    name : "shane",
    age : 20,
    marks : 99.9,
    city : "chennai"
};
```

- Change the city to "Kerala".
- Add a new property, gender : "Male"
- change the marks to "A".

```
student.city = "Kerala";
student.marks = "A";
student.gender = "Male;
```

To delete

```
delete object.key;

delete student.age;   // To delet student age.
```

Day 33 of 100 (Sunday) 09/02/2025

Nested Objects

Object of Objects

```
const classinfo = {
    shane : {
        age : 20,
        marks : [99, 100, 99]
    },
    sven : {
        age : 22,
        marks : [100, 99, 100]
    }
};
```

```
classinfo.shane
classinfo.shane.marks
classinfo.shane.marks[0]
```

Array of Objects

```
const class info = [
    { name : "shane",
      grade : "O",
      city : "chennai"
    },
    { name : "sven"
      grade : "O",
      city : "Newyork"
    }
];
```

Math object

Properties	Methods
Math . PI	Math . abs(n)
Math . E	Math . pow (a,b)
	Math . floor(n)
	Math . ceil (n)
	Math . random ()

gives absolute value (+ve)

a^b

nearest smallest integer

nearest largest integer.

Random Integers

Math . random () ∈ (0 to 1)

(Math . random()) * 10 = (0 to 10 including decimal)

math . floor((math . random ()) * 10)

0 to 9 only integers.

Day 34 of 100 (Monday) 10/02/2025

Functions in JS

console.log ()

function Definition

```
function functionName {
    //do something
}
function Calling (using the function)
  func Name();
```

eg:-

```
function hello{
console.log ("Hello");
}

hello();
```

Function with Arguments

Values we pass to the function.

```
function funcName (arg1, arg2 ---) {
    //do something
}

function printInfo (name, age) {
 console.log (name, "is", age, "years old");
}
```

Return

Return keyword is used to return some value from the function.

input → Function (does some work) ⟶ output

```
function funcName (arg1, arg2 ---) {
    // do something
    return val;
}
```

Day 35 of 100 (Tuesday) 11/02/2025

Scope

Scope determines the accessibility of variables, objects and functions from different parts of the code.

- Function
- Block
- Lexical

* Global

Function Scope

Variables defined inside a function are not accessible (visible) from outside the function.

Block Scope

Variables declared inside a { } block cannot be accessed from outside the block.

Lexical Scope

A variable defined outside a function can be be accessible inside another function defined after the variable declaration.

The opposite is not true.

```
fun1{
let x = 1;
fun2{
console.log(x);
}
```

hoisting

Function Expressions

```
const variable = function (arg1, arg2 --)
{
  // do or return something.
}
```

```
const sum = function (a,b) {
  return a+b;
}
sum(2,3);
```

High Order Functions

A function that does one or both:

- takes one or multiple functions as arguments
- returns a function.

One or More Functions as arguments

```
function multipleGreet (func, n) {
    for (let i=1; i<=n ; i++) {
        func();
    }
}

let greet = function () {
    console.log ("hello");
}

multipleGreet (greet, 2);
```

Higher order Functions

Returns a function

```
function oddeven (req) {
    if (req == "odd") {
        return function (n) {
            console.log(!(n%2 == 0));
        }
    } else if (req == "even") {
        return function (n) {
            console.log (n%2 == 0);
        }
    } else {
        console.log ("wrong request");
    }
}
```

Methods
Actions that can be performed on an object

```
const calculator = {
    add : function (a,b) {
        return a+b;
    },
    sub : function (a,b) {
        return (a-b);
    },
    mul function (a,b) {
        return a*b;
    }
};
```

Day 36 of 100 (Wednesday) 12/02/2025

Javascript (Part 7)

'this' key word
"This" keyword refers to an object that is executing the current piece of code.

```
const student = {
    name : "shane",
    age : 20,
    eng : 95,
    math : 93,
    phy : 97
}
```

Day 36 of 100 (wednesday) 12/02/2025

"this" keyword

```
const student = {
    name: "Shane",
    age : 20;
    eng : 96,
    math : 100,
    get Avg() {
        let avg = (this.eng + this.math)/2 ;
        console.log (avg);
    }
}
```

"This" keyword refers to an object that is executing the current piece of code.

try & catch

The try statement allow you to ~~create~~ define a block of code to be tested for errors while it is being executed.

The catch statement allow you to define a block of code to be executed, if an error occurs in the try block.

Miscellaneous Topics

Arrow Functions (without writing "function" keyword)

```
const func = (arg1, arg2 ---) => { function
                                   defenition }
```

eg
```
const sum = (a,b) => {
        console.log (a+b);
}
```

For functions with no arguments

```
const wish = () => {
    console.log ("Hello");
}
```

* Implicity return

~~const mult = (a,b) =>~~

```
const mult = (a,b) => (
    a*b
);
console.log (mult(9,4));
```

no semicolon.

Day 31 of 100 (Thursday) 13/02/2025

Set TimeOut Function (delay) only for particular function

set Timeout (function, timeout)

console.log ("Hello!");

~~set Timeout~~
setTimeOut (() => {
    console.log ("Shane Sunny Kottuppallil");
3, 4000);

console.log("welcome. I am ");

setInterval (repeat after each interval)

setInterval (function, timeout)

setInterval(() => {
    console.log ("Shane Sunny");
3, 2000);

clearInterval (id)

"this" Keyword with Arrow Functions

Arrow functions

1) Lexical scope
(parent → call scope)

eg: object {
function ⇒ (this) } ⇒ parent is object.
}

eg function (this) } parent is global.

~~Parent of object is scope~~

Normal function

1) Scope ⇒ this → calling object.

object {
function (this)
}

Object is the scope

→ Parent of Object's scope (global)

For Normal function
→ "this" is the one who called the function.

Day 38 of 100 (Friday) 14/02/2025

Array Methods

- forEach
- map
- filter
- some
- every
- reduce.

forEach (does some function to each elements)

arr.forEach (some function defenition or name);

```
let arr = [12, 5, 9, 17, 20];
let print = function (el) {
    console.log (el);
}
arr.forEach (print);
```

output
12
5
9
17
20

Map & Filter

Map (maps new values to new array)

let newArr = arr.map (some function defenition or name)

```
let num = [1, 2, 3, 4];
let double = num.map (function (el) {
    return el*2;
});
```

output ⇒ 2, 4, 6, 8

Filter

let newArr = arr.filter (some function defenition or name);

```
let nums = [2, 4, 1, 5, 6, 2, 7, 8, 9];
let even = nums.filter ((num) => (num % 2 == 0));
```

output ⇒ 2, 4, 6, 2, 8

Every

Returns true if every element of array gives true for some function. Else returns false.

arr.every (some function defenition or name);

let arr = [1, 2, 3, 4, 5, 6]

arr.every((el) => (el % 2 == 0));
=> false

Some

Returns true if some elements of array give true for some function. Else returns false.

arr.some (some function defenition or name);

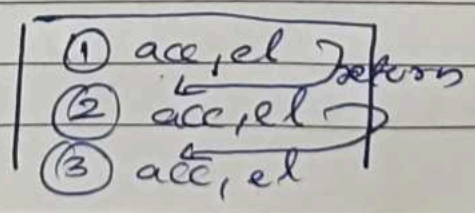
[1, 2, 3, 4].some ((el) => (el % 2 == 0));
=> true.

Reduce

Reduces the array to a single value.

arr.reduce (reducer function with 2 variables for (accumulator, element));

[1, 2, 3, 4].reduce ((res, el) => (res + el));
=> 10



~~Finding Maximum in an array~~

Finding Maximum of an array using reduce

```
let arr = [5, 6, 1, 7, 10, 12, 3, 4, 1];
let max of Array = arr.reduce((max, el) => {
        if(max > el){
          return max;
        }else {
          return el;
        }
});
console.log(max of Array);
```

Default Parameters

Giving a default value to the arguments.

```
function func (a, b = 2) {
  //do something
}

function sum (a, b = 3) {
        return a+b;
}
sum(2); //5
```

Day 39 of 100 (Saturday) 15/02/2025

Spread
Expands an iterable into multiple values

```
function fun (...arr) {
    // do something.
}
```

... ⇒ passes values one by one.

Spread with Array Literals

```
let arr = [1,2,3,4,5];
let newArr = [... arr];
newArr;
⇒ (5) [1,2,3,4,5]
```

```
let chars = [..."hello"];
chars
⇒ (5) ['h', 'e', 'l', 'l', 'o']
```

```
let arr1 = [1,2,3,4,5];
let arr 2 = [6,7,8,9,10];
let arr 3 = [...arr1, ...arr2];
console.log (arr3);
⇒ [1,2,3,4,5,6,7,8,9,10];
```

Spread with Object Literals

```
let data = {
    email : "shanesunny ssk@gmail.com",
    password : "password";
};
let data copy = {...data, id : 123};
```

Array to object

```
let arr = [ 12, 13, 15, 20, 9, 25];
let obj = { ... arr};
console . log (arr);
console . log (obj);
```

⇒ [12, 13, 15, 20, 9, 25];
{ 0: 12, 1: 13, 2: 15, 3: 20, 4: 9, 5: 25 }

Index of array → 0
Value of array → 12

new object (Key/Value Pair)

Rest

Allows a function to take an indefenite number of arguments and bundle them in an array.

```
function sum (... args) {
   return args. reduce ((add, el) ⇒ add + el);
}
```

Destructing (Arrays)

Storing values of array into multiple versions.

```
let names = ["shane", "sven", "shalets" "sunny"];
let [winner, runnerup] = names;
console. log (winner, runnerup);
```

⇒ "shane" "sven"

Destructing (Objects)

```
const student = {
    name : "shane",
    class : 12,
    age : 18,
    subjects : [ "english", "physics", "chemistry", "maths",
                 "computer science"],
    username : "shanesunnyssk",
    password : "password"
};
```

```
const {username : user, password : pass} = student;
console.log (user); // shanesunnyssk
```

Practice Question 4

Write a function ~~called double~~ which accepts array and a variable number of arguments. The function should return a new array with original array values and all of the additional arguments obtained.

```
let randarr = [1,2,3,4,5];
let newrandarr = [6,7,8,9,10];
let final Array = function (a,b) {
  let newb = b.map((el) => el*2);
  finarr = [...a, ...newb];
  return finarr;
};
console.log (final Array (randarr, newrandarr));
console.log ("\n");
```

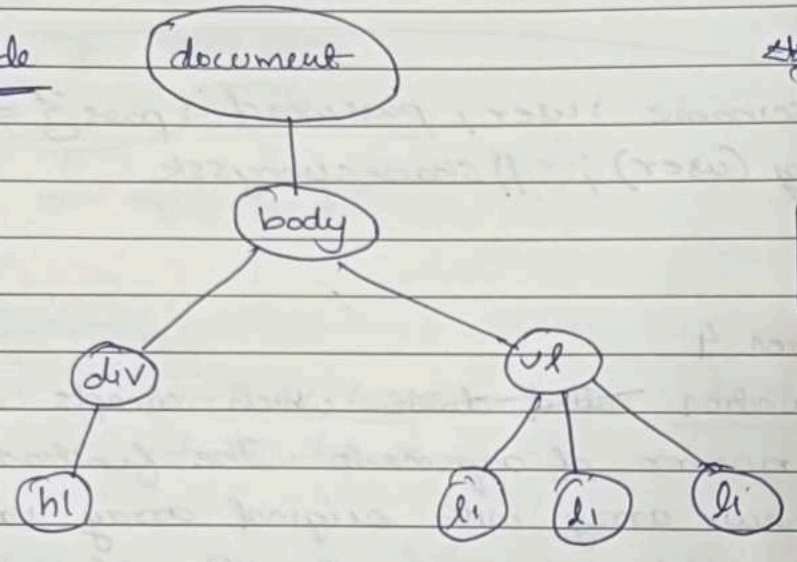
Day 40 of 100 (Sunday) 16/02/2025

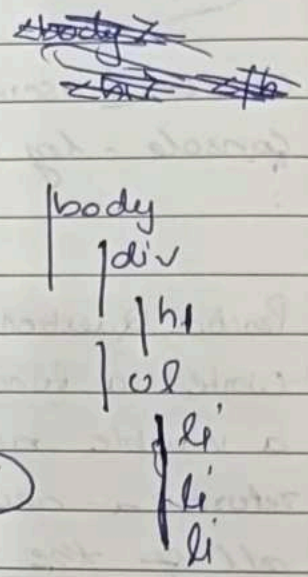
DOM (Document Object Model)

The DOM represents a document with a logical tree. It allows us to manipulate / change webpage content (HTML elements)

DOM

node





console.dir(document); ⇒ to view document object.

1) Select elements
2) Changes / manipulation.

Selecting Elements

1) getElementById
Returns the Element as an object or null (if not found)

if id is not present then it will return null

innerText() ⇒ to manipulate innertex of paragraph, header etc.

90

⇒ document.getElementById(id);

2) getElementsByClassName

Returns the Elements as a HTML collection or empty collection (if not found)

⇒ document.getElementsByClassName(class-name);

If class is not present then it wont show null it will return HTMLcollection [] which is empty.

3) getElementByTagName

Returns the Element as an HTML collection or empty collection (if not found)

Query Selector

Allows us to use any CSS selector.

```
document.querySelector('p'); // selects first p element.
document.querySelector('#myId'); // selects first element
                                    with id = myId.
document.querySelector('.myClass'); // selects first
                                    element with class = myClass.
document.querySelectorAll('p'); // selects all p elements.
```

pseudo selector

document.querySelector('.className:nth-of-type(2n+1)

Query Selector selects only a single element

Node list ⇒ Query SelectorAll selects all elements specified.

Day 41 of 100 (Monday) 17/02/2025

Setting Content in Objects

Using Properties & Methods

innerText
shows the visible text contained in a node.

textContent
Shows all the full text.

innerHTML
shows the full markup. (including hidden content)

eg:- let heading = document.querySelector('h1');
heading
⇒ <h1> Spider man </h1>
heading.innerHTML = `<u> ${heading.innerText} </u>`;

`` ⇒ back tick (not '' or "")

background-image ⇒ CSS
↓ camelCase
backgroundImage ⇒ JS

Manipulating Attributes

obj.getAttribute(attr)
obj.setAttribute(attr, val)

getters & setters

attributes in HTML
#id
.class
style
<src>

```
let img = document.querySelector('img');
img.getAttribute('id');
```
⇒ main Img
```
img.setAttribute('id', 'spiderman(mg');
```

Manipulating Style

style Property (only for inline styles)
obj.style

classList Property
using classList

obj.classList

classlist.add() ⇒ to add new classes.
classlist.remove() ⇒ to remove classes.
classlist.contains() ⇒ to check if class exists.
classlist.toggle() ⇒ to toggle between add & remove.

toggle ⇒ to add and remove classes.

~~Navigation~~
~~parent Element~~

Navigation
parentElement
children
previousElementSibling / nextElementSibling.

childElementCount.

Adding Elements

document . createElement ('p')

- appendChild (element) } adds element in the last
- append (element)
- prepend (element) — adds element in the start.
- insertAdjacentElement (where, element)

eg body . appendChild ('btn') ; ⇒ <body>
<btn>
</body>

append (element)
element or string.

prepend(element)

insertAdjacentElement(position, element)

positions
'before begin'
'after begin'
'beforeend'
~~'befor~~
'afterend'

Removing Elements

- remove child(element);
- ~~remove(element)~~ element.remove();

Day 42 of 100 (Tuesday) 18/02/2025 (JS Part 10)

DOM Events
Events are signals that something has occured (user inputs/actions)

button ⇒ onclick (trigger)

onclick() ⇒ when an element is clicked.
onmouseenter() ⇒ when mouse enters an element.

~~Event listeners~~

regular function ⇒ function () { }
arrow function ⇒ () ⇒ { }

Event listener for elements

addEventListener

element.addEventListener(event, callback)

eg:-
```
btn.addEventListener("click", function() {
    console.log("button clicked");
});
```

'this' in Event listeners

when 'this' is used in a callback of event handler of something, it refers to that something.

make sense to use (function () { }) and not (() ⇒ { }) for proper functioning

```
let newbtn = document.querySelector(".button");
newbtn.addEventListener("click", function () {
    console.log(this);
    console.dir(this);
})
```

Day 43 of 100 (Wednesday) 19/02/2025

JS Part10 (conti--)

Keyboard Events

code:
key:

event.preventDefault() ⇒ to prevent default actions
used to prevent form action.

input.value

event
- code → gives code of the letter (or) key.
- key → what is visible on screen.

event.key
event.code

A
event.key = a
event.code = KeyA

Form events

```
let form = document.querySelector("form");
form.addEventListener("submit", function () {
    console.log("Form Submitted");
});
```

Extracting form Data

```
let body = document.querySelector("body");
let form = document.querySelector("form");
form.addEventListener("submit", function () {
    event.preventDefault;
    console.log("Form Submitted");
    let inp = document.querySelector("input");
    let p = document.createElement("p");
    p.innerHTML = `Your Name is ${inp.value}`;
    body.appendChild(p);
});
```

event.stopPropagation();

forms.element[0]

More Events

change event (if any change is made and clicked somewhere)
The change event occurs when the value of an element has been changed (only work on <input>, <text area>, and <select> elements).

input event (only when submit)
The input event fires the value of an <input>, <select> or <text area> element has been changed.

Day 43 of 100 (wednesday) 19/02/2025
JS Mini-Project

Event Bubbling

```
<div>
    <ul>
        <li>    </li>
        <li>    </li>
        <li>    </li>
    </ul>
</div>
```

event.stopPropagation() ⇒ to stop bubbling

Todo list

Event Delegation

event.target ⇒ <button> delete </button>

event.target.nodeName ⇒ button

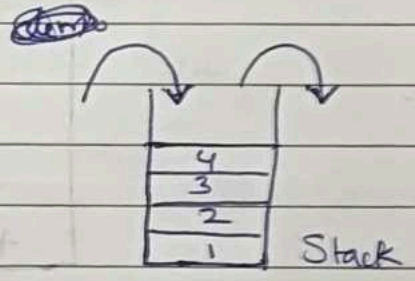
Day 44 of 100 (Thursday) 20/02/2025

Simon Says Game

Day 45 of 100 (Friday) 21/02/2025

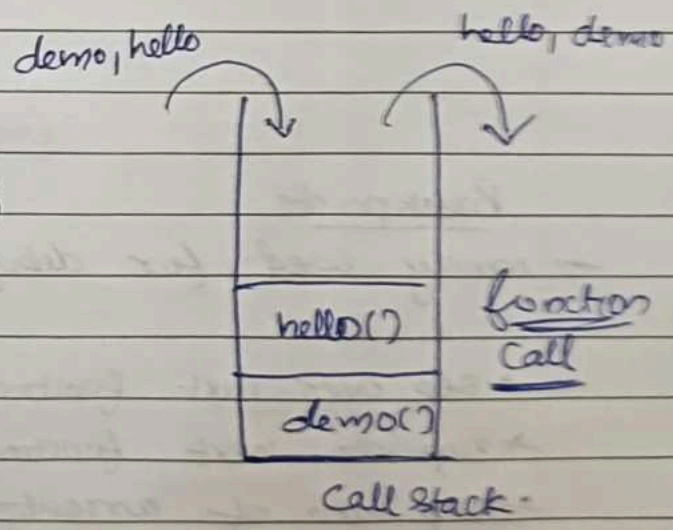
Call Stack

+ Stack follows LIFO
Last in First out.



eg:-

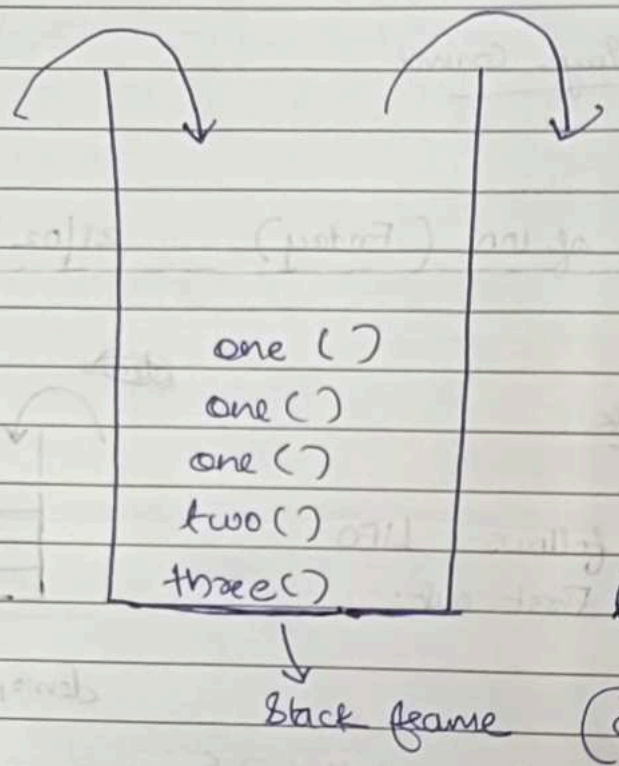
```
function hello() {
  console.log("hello");
}
function demo() {
  hello();
}
demo();
```



Visualizing the call stack

eg2

```
function one() {
  return 1;
function two() {
  return one() + one();
}
function three() {
  let ans = two() + one();
  console.log(ans);
}
three();
```



Breakpoints

* mainly used for debugging.

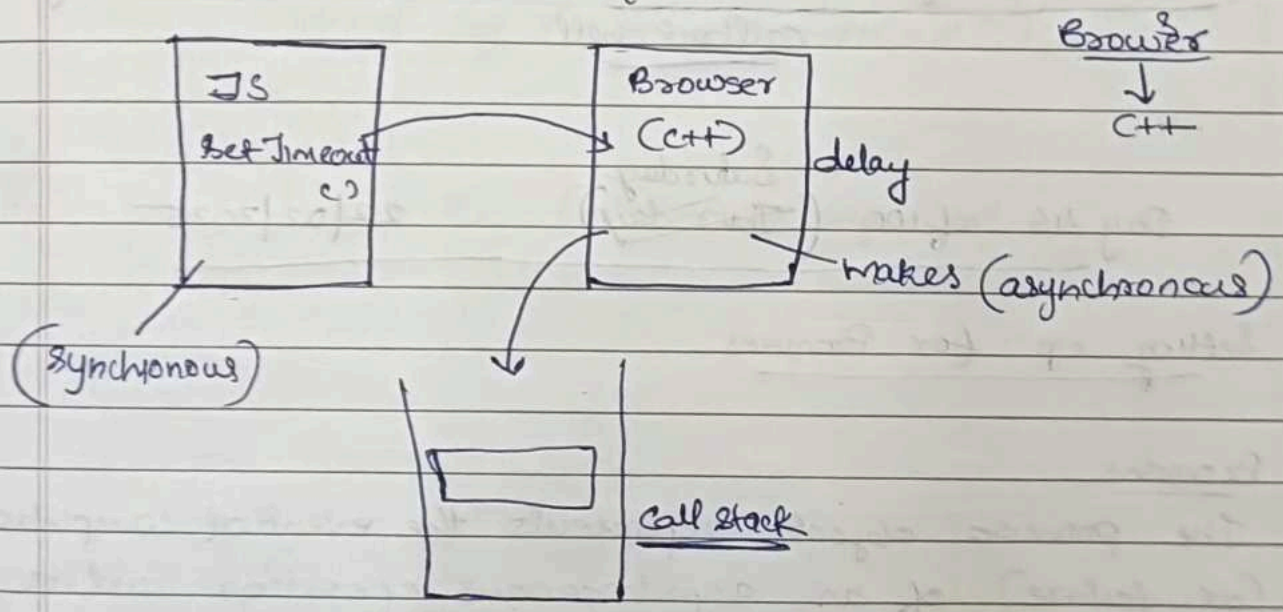
* step over next function call
* step into next function call ✓
* step out of current function

JS is Single Threaded (Synchronous)

```
setTimeout (function () {
    console.log (" apna college ");
3, 2000);
console.log ("hello --- ");
```

Single threaded ⇒ a program executes one instruction at a time.

Callbacks ⇒ a function that is passed as an argument to another function so that it can be executed in that other function.



~~Callback Hell~~

Callback Hell

```
function changeclr (color, delay , nextclrchange) {
    setTimeout (() => {
        h1.style.color = color;
        if(nextclrchange) nextclrchange();
    }, delay);
}

changeclr ("red", 1000 , () => {
    changeclr ("orange", 1000, () => {
        changeclr ("green", 1000);
    });
});
```

} Callback Hell

Callback Nesting

promises, async & await used for preventing callback hell.

Day 46 of 100 (~~Thursday~~ Saturday) 22/02/2025

Setting up for Promises

Promises

The promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

* resolve & reject
  - resolve → success
  - reject → failure

promise states

* pending
* rejected ⇒ failure
* fullfilled ⇒ success

promise methods

then() & catch()

then() ⇒ to do some task after fullfilled
catch() ⇒ to do some task for reject

```
let request = saveToDB("Shane Sunny");
request
    .then(() => {
        console.log("promise resolved");
    })
    .catch(() => {
        console.log("promise rejected");
    });
```

Promise Chaining
Improved Version

eg:-
```
saveToDBPromise("Shane Sunny")
    .then(() => {
        console.log("promise resolved");
        return saveToDBPromise("hello world");
    })
    .then(() => {
        console.log("promise 2 resolved");
    })
```

```
.catch (() => {
    console.log ("Some promise rejected");
});
```

* promises are rejected and resolved with some data (valid results or errors)

eg:-
```
saveToDBPromise ("Shave Sunny")
  .then (result) => {
    console.log ("result : ", result);
    console.log ("promise 1 resolved");
    return saveToDBPromise ("hello world");
  })
  .then ((result) => {
    console.log ("result :", result);
    console.log ("promise 2 resolved");
  })
  .catch(error) => {
      console.log ("error : ", error);
      console.log ("some promise rejected");
  });
```

Day 47 of 100 (Sunday) 23/02/2025

JS Part 12

async & await Keywords

Async Keyword

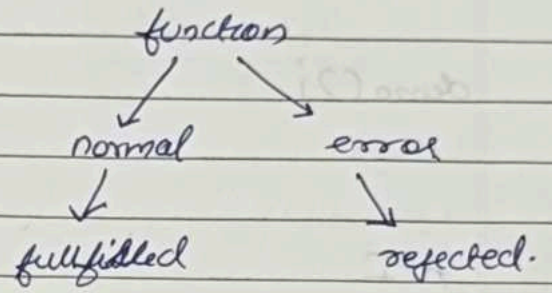
Creates an Async Function.

eg:-

Async function ⇒ returns promise
"throw" keyword ⇒ throws error.

eg:-

```
async function greet () {
    return "hello world!" ; //returns a promise
}
let hello = async () => {} ;//returns a promise
```



Await Keyword

pauses the execution of its surrounding async function until the promise is settled (resolved or rejected)

```
async function show () {
    await colorChange ("violet", 1000);
    await colorChange ("indigo", 1000);
    await colorChange ("green", 1000);
    await colorChange ("yellow", 1000);
    return "done";
}
```

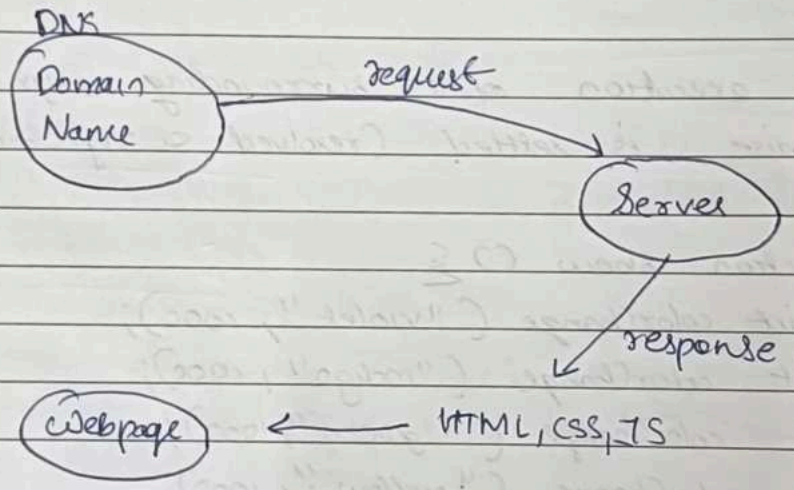
Handling Rejections with Await

```
async function demo () {
    try {
        await changeClr ("red", 1000);
        await changeClr ("yellow", 1000);
    }
    catch (err) {
```

```
    console.log("error caught");
}
~~console.log~~
let a=5;
console.log(a);
console.log("new number:", a+3);
}

demo();
```

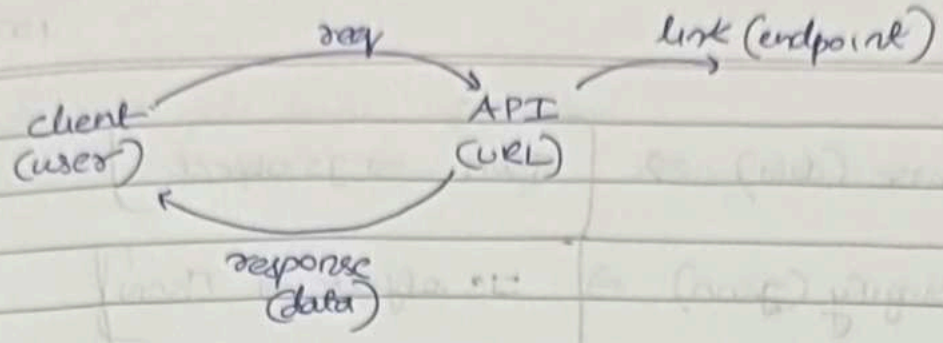
API



Application Programming Interface.

S/W ⟷ API ⟷ S/W WebAPI

APIs
↓
data
(JSON)



JSON (JavaScript Object Notation)

eg

```
{
    "string" : "Hi",
    "number" : 2.5,
    "boolean" : true,
    "null" : null,
    "object" : {"name" : "Kyle", "age" : 24},
    "array" : ["Hello", 5, false, null, {"key" : "value",
                "number" : 6}],
    "arrayofobjects" : [
        {"name" : "Jerry", "age" : 28},
        {"name" : "Shane", "age" : 26}
    ]
}
```

Accessing Data from JSON

* JSON.parse(data) Method

To parse a string data into a JS object.

* JSON.stringify(Json) Method

To parse a JS object data into JSON.

- JSON.parse (data) ⇒ [Json ⇒ JS object]
- JSON.stringify (Json) ⇒ [JS object ⇒ JSON]

API Testing Tools

- Hoppscotch
- Postman

Day 48 of 100 (Monday) 24/02/2025

JS Part 12 (Part 2)

What's Ajax?

Ajax
Asynchronous JavaScript and XML

XML ⇒ Extensible Markup language.

JS —req→ API ⇒ Asychronous
JS ←res— API

HTTP Verbs

eg:- • GET
• POST
• DELETE

GET ⇒ to get some information
POST ⇒ to send some info through API calls

Status Codes

200 – OK
404 – Not Found
400 – Bad request
500 – Internal Server Error.

Add information in URLs

https://www.google.com/search?q=harry+potter
key value

?name = Shane & marks = 100

url/:id
:num
:q

{{ id }}
<id>

HTTP Headers (header, value)

Response Header
Request Header.

Request Header
Authority
Method
path

To get JSON

header name	Value
eg Accept	application /JSON.
eg Accept	text/plain.

First API Request

using Fetch {XMLHTTPRequestobj}

fetch(url)

fetch() ~~ret~~ returns a promise.

• JSON

```
eg- let url = "          "
    fetch(url)
    • then((res) => {
        return res • Json();
    })
    • then(data => {
        console . log (" data1 = ", data );
        return fetch(url);
    })
    • then((res) => {
```

```
    return res.json();
  })
  .then((data2) => {
      console.log("data2 = ", data2);
  })
  .catch((err) => {
      console.log("error: ", err);
  });
```

* <u>using fetch with async ~~await~~</u>

eg:- let url = " "

~~fetch~~

```
async function getFacts() {
    try {
        let res = await fetch(url2);
        console.log("Fetch Promise : ", res);
        let validData = await res.json();
        console.log("Value of Second Fact: ", validData.fact);
    }
    catch {
        console.log("Error in fetching url");
    }
}
getFacts();
```

Day 49 of 100 (JS Part-13) 25/02/2025
(Tuesday)

Axios
Library to make HTTP requests.

eg:-

```
axyx function getFacts () {
    try {
        let res = await axios.get(url);
        console.log (res);
    } catch (e) {
            console.log ("Error :", e);
    }
}
```

Axios (Sending Headers)

```
const config = { headers : { Accept : "application /json" } }
let res = await axios.get (url2, config);
console.log (res.data);
```

headers
eg application /json
plain /text

~~Day 50 of 100~~

Axios (Updating Query Strings)

* let url = " /search? = "
* axios.get(url + country);
  search

~~Day 50 of 100~~

Day 50 of 100 26/02/2025 (Wednesday)

Terminal

What is Terminal?
* A text input and output environment.

1) Easy to use
2) Speed
3) Access

Different Terms

- Command Line :- any interface that is used by entering textual commands.
- Terminal :- This is a type of command line
- Console :- A command-line interface used to work with your computer.

- shell – A program running on terminal.
- Bash – A popular shell on Linux/Mac OS.
- Z-Shell – Another shell.

Basic Commands

pwd ⇒ working directories (print)
clear ⇒ to clear screen data
git --version ⇒ shows git version
ls ⇒ lists folders in the directory
↓
& files

~ ⇒ Home Directory.

Navigation
Inside and Outside Directories

cd = change directory
cd ..
↑ space
⇒ back

Tab ⇒ to auto complete

cd ../.. ⇒ 2 steps back.

Paths in Navigation

Absolute & Relative

- cd Desktop/Shane (Relative path)
  cd /Users/Shane Sunny/Desktop (absolute path)

absolute path ⇒ starts with a '/' slash

/ ⇒ root directory
~ ⇒ home directory

Making Directories

mkdir = make directory

Flags
flags are characters that we pass with commands to modify their behaviours.

manual command

man -ls ⇒ gives info about ls commands
man -mkdir ⇒ gives info about mkdir command.

with Flags

ls -l
ls -a
ls -la

Touch Command
Used to create files.

touch index.html
touch app.js
touch abc.txt

* It helps to change file access and modification time.

Highly Sensitive Commands

Deleting Files & Folders (Non-Recyclable)

rm ⇒ remove files
rmdir ⇒ remove empty folders
rm -rf ⇒ removes any folders

-r ⇒ recursive delete
-f ⇒ forced delete
⇒ -rf

Day 51 of 100 27/02/2025 (Thursday)

Git & Github

Git
Free & Open Source Version Control System
tool that help to track changes in code.

1) Track history
2) helps to collaborate.

Github
Website where we host repositories online.

* Creating Repositories
public / private Repo

README.md .md ⇒ markdown

commit ⇒ change.

add → commit

Using Git

- Command Line
- IDE/Code Editors (Like VSCode)
- Graphical User Interface (like GitKraken)

// git --version

Configuring Git

```
git config --global user.name "My Name"
git config --global user.email "someone@email.com"
```

--local
--system
--global

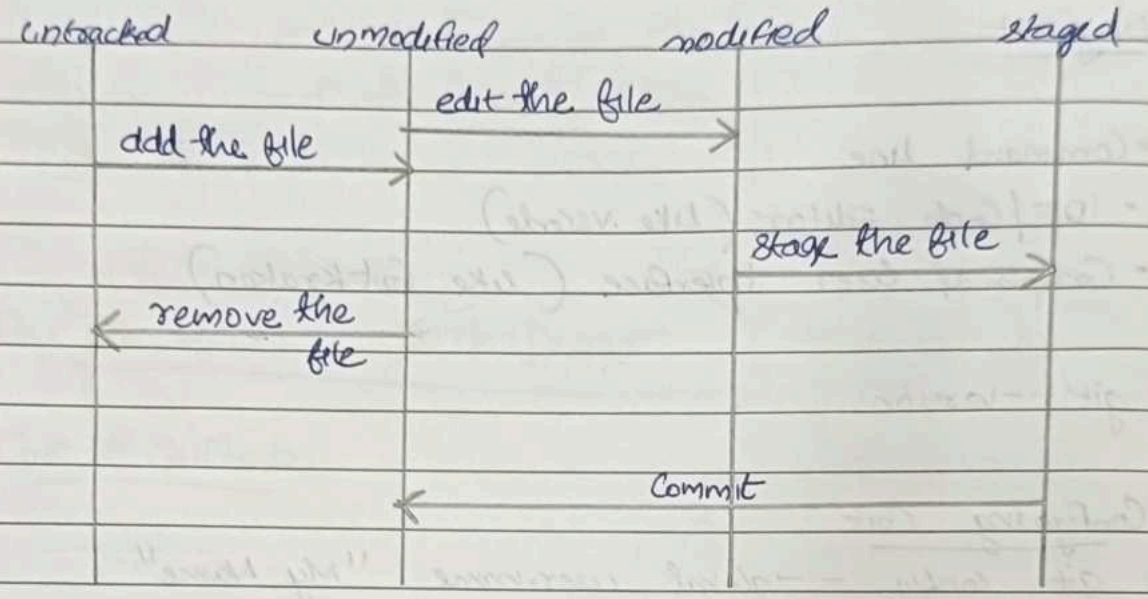
```
git config --global user.name "Shane Sunny Kotteppalli"
```

```
git config --list
```

Basic Commands

- Clone :- Cloning a repository on our local machine.
- Status :- displays the state of the code.

File Status lifecycle



- git clone link
- git status

Add & Commit Commands

add – adds new or changed files in your working directory to the Git staging area.
commit – it is the record of change.

push – upload local repo content to remote repo

- git add <-filename->
- git commit -m "some message"
- git push origin main

• ⇒ all
-m ⇒ message flag

⇒ git add • ⇒ to add all

⇒ git push origin main.
(branch name.

Day 52 of 100 28/02/2025 (Friday)

Git & Github (Part 2)

Init Command

• init – used to create a new git repo
⇒ git init

git remote add origin <-link->
git remote -v (to verify remote)
git branch (to check branch)
git branch -M main (to rename branch)
git push origin main

git push -u origin main ⇒ to make default origin main

add and commit
git commit -am "Initial"
|
add & commit

Workflow

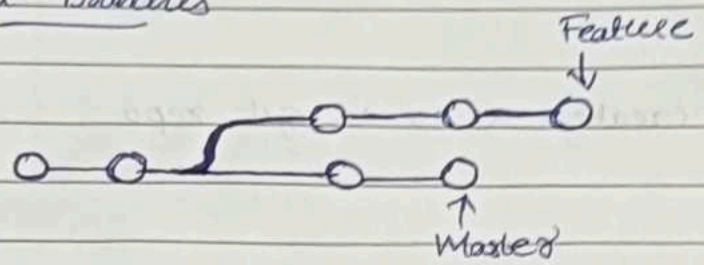
Github

code changes
↓
commit

Local Git

↓
code changes
↓
add
↓
commit
↓
push

Git Branches



Branch Commands

git branch (to check branch)
git branch -M main (to rename branch)

git checkout <-branch name-> (to navigate)

git checkout -b <-new branch name-> (to create new branch)

git branch -d <-branch name-> (to delete branch)

Merging branch

git push --set-upstream origin <-branch name->
to push a branch

Merging Code

- git diff <-branch name-> (to compare commits, branches, files & more)

- git merge <-branch name-> (to merge 2 branches)

(or)

Create a Pull Request.

Pull requests
It lets you tell others about change you've pushed to a branch in a repo on github.

- git pull origin main
  used to fetch and download content from a remote repo and immediately update the local repo to match that content.

Merge Conflicts
An event that takes place when Git is unable to automatically resolve differences in code between two commits.

Fixing Conflicts

Case 1: staged changes
  get reset <-file name->
  get reset

git commit -am "commit message" ⇒ add + commit

git log (log of all commits)

Case 2: Commited changes (for one commit)

git reset Head ~1

Case 3: Commited changes (for many commits)

git reset <- commit hash ->

git reset --hard <- commit hash ->

→ to even remove any modification

→ brings back to modified.

Forking

Fork is a new repository that shares code and visibility settings with the original "upstream" repository.

Fork is a rough copy.

Day 53 of 100 March 01/03/2025 (Saturday)

Backend! (Node.JS)

Node.JS

JavaScript Runtime Environment

* It is used for server side programming

* Node.JS is not a language, library or Framework.

node -v ⇒ to check node version.

Node REPL

- help gives us commands

Read - Evaluate - Print - Loop.

global object ⇒ similar to window object on browser

- exit ⇒ to exist Node environment.

Node files

node fileName.js ⇒ to run Javascript files on Node

Process in Node

process : This object provides information about, and control over, the current Node.JS process.

process.argv : returns an array containing the command-line arguments passed when the Node.JS process was launched.

process.cwd() ⇒ current working directory.

Day 54 of 100 02/03/2025 (Sun ~~Tuesday~~)

Export in Files

requiring files

require() → a built-in function to include external modules that exist in separate files.

module.exports ⇒ a special object

module.export ⇒ to export functions, codes from one JS file to another.

Export in Directories

requiring directories

require() ⇒ a built-in function to include external modules that exist in separate files.

module.exports ⇒ a special object.

1) require → index.js → all exports.
(entry point)
similar to main class in C, C++, Java

NPM (Node Package Manager)

npm is the standard package manager for Node.JS

* library of packages
* command line tool.

npm

eg:- npm i flylet
alias

Installing packages

npm install <- package name ->

node_modules :- The node_modules folder contains every installed dependency of your project.

package-lock.json :- It records the exact version of every installed dependency, including its sub-dependencies and their versions.

package.json
The package.json file contains descriptive and functional metadata about a project, such as a name, version and dependencies.

npm init

Local v/s Global

npm install -g <- package name ->
npm link <- package name ->

sudo chown -R $USER /usr/local/lib/node_modules

require v/s import

import {sum} from "./math.js"

We cant selectively load only the pieces we need with require but with import, we can selecti

vely load only the pieces we need, which can save memory.

Loading is synchronous for 'require' but can be asynchronous for 'import'.

math.JS
```
export const sum = (a,b) => a+b;
```

script.JS
```
import {sum} from "./math.JS"
```

package.Json
```
{
"type": "module"
}
```
include this in the package.Json of the directory where math.JS and script.JS is present.

Day 55 of 100 03/03/2025 (~~Wednesday~~ Mon)

Backend 2 (Node : Express)

Library	Framework
A library is a collection of pre-written code that can be used fo	A framework is a set of pre-written code that provides a structure for developing

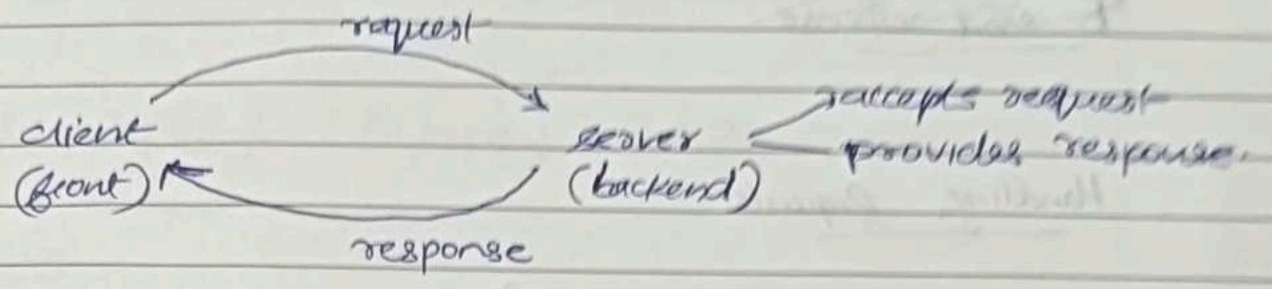
axios :- sending headers.

perform specific tasks eg :- axios.	software applications eg :- React, Express

Express

A Node.js web application framework that helps us to make web applications.

It is used for server side programming.



1. listen for incoming request.
2. parse
3. Match response with routes.
4. response

Getting Started with Express

```
const express = require ("express");
const app = express()
let port = 8080;

app.listen (port, () => {
  console.log( `app listening on port ${port}` );
});
```

Ports are logical endpoints of a network connection that is used to exchange information between a web server and a web client.

Day 54 ⇒ cd Day 54

3000/8080 ⇒ available ports for custom requests.

app.listen ⇒ listens incoming request.

```
app.listen (port , () => {
    console.log ('app is listening');
});
```

⇒ localhost : 3000

~~To send response~~

Handling Requests

app.use (To send response)

eg:-
```
app.use((req, res) => {
    console.log ("new incoming request");
});
```

Sending a Response

request (req) → object

response (res)

res.send () ⇒ Sends the HTTP response.

* Express converts JS object to JSON on

'*' selects all path (except exceptions)

Sending a response.

~~Routing~~
~~It is process of selecting a path for traffic in a network or between or across multiple network~~

Day 56 of 100 04/03/2025 (~~Thursday~~ Tuesday)

Backend 2 (Node : Express) Part 2

Routing
It is process of selecting a path for traffic in a network or between or across \ multiple networks

```
app.get ("/apple", (req, res) => {
    res.send ({
        name : "apple",
        color : "red",
    });
});
```

app.get (path, callback);

Nodemon
To automatically restart server with code changes

nodemon -v

execution
nodemon <- file name . type ->

Path Parameters

req.params

```
app.get ("/ig/:username", (req, res) => {
    let {username} = req.params;
    res.send ('This account belongs to @${username}');
});
```

/:

Query Strings

req.query

```
app.get ("/search", (req, res) => {
    let {q} = req.query;

    if(!q) {
        res.send ("No search query");
    }
    res.send ('These are the results for : ${q}');
});
```

Templating :- EJS, Handlebar

npm init -y
↳ flag to automatically fill options.

Day 57 of 100 05/03/2025 (~~Friday~~ wednes)

Backend 3 (Node : EJS)

Jemplating
EJS (Embedded Javascript templates)
EJS is a simple templating language that lets us generate HTML markup with plain Javascript.

npm init -y
npm install ejs

Using EJS

```
app.set ("view engine", "ejs");

app.get ("/", (req, res) => {
    res.render ("home.ejs");
});
```

view ⇒ template

ejs dont send response, while it renders the response.

views directory ⇒ default directory for ejs files to be stored for express to retrieve them.

~~app.set ("views", path.join (__dirname~~

res.render ("ejs file", "object")
↓
data

Views Directory

To access views directory from any folder to run an index.js file and get home.ejs

⇒ app.set ("views", path.join (__dirname, "/views"));

for this

const path = require ("path"); 1st

Interpolation Syntax

Interpolation refers ~~to~~ to embedding expressions into markedup text.

* EJS can be used to make our ~~HTML~~ HTML dynamic.

ejs.co

* Tags

<%= %>

Outputs the value into the template (HTML escaped)

Passing data to EJS

~~app.get~~

```
app.get ("/roll dice", (req, res) => {
    let num = Math.floor (Math.random ()*6)+1
    res.render = ("rollDice.ejs", { diceVal : num })
});
```

object

```
<h1> Your dice gave value : <%= diceVal %> </h1>
```

Instagram EJS

Create a basic template for instagram page based on following route:

/ig/:username

Conditional Statements

Adding conditional statements inside EJS.

```
<%= if (diceVal ==6) { %>
<h2> Nice! Roll dice again. </h2>
<% } %>
```

Loops for EJS

```
<ul>
<% for(name of followers) { %>
  <li> <%= name %> </li>
<% } %>
</ul>
```

Day 58 of 100 06/03/2025 (Thurs day)

Backend 3 (Node: EJS) Part 2

```
const instaData = require("./data.json");
```

Serving Static Files (~~HTML~~, CSS, JS)

```
app.use(express.static(folder_name))
```

express.static (folder_name) ⇒ middleware

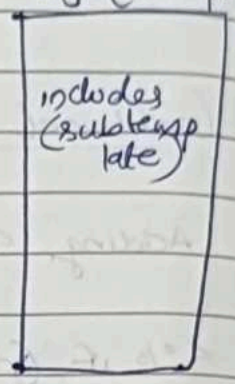
app.use (express.static (path.join (__dirname, "public")))

Includes (partials) (subtemplates)

<%- include ("includes/head.ejs"); %>

no slash

ejs (templates)



<%- Outputs the unescaped values into the template.