HTML

```
<!DOCTYPE HTML>
<html>  → (root element)
  <head>
    <title> Page Title </title>
  </head>
  <body>
    <h1> This is a heading. para </h1>
    <p> This is a para
  </body>
</html>
```

- Boiler plate / Structure of HTML
- This is HTML Boilerplate Contain the nested html elements

- HTML → Hyper Text Markup Language.
- Standard Markup language for creating web pages

<!DOCTYPE> → This declares that this is the document of an html element

<h1> → Define the large heading

<tagname> Content goes here .... </tagname> HTML element

↳ from the starting tag to ending tag

<h1> </h1> → heading.

  → Paragraph

 → break with no closing tags

<body>

/. Everything which needed to be written in the html document will comer under the body.

</body>

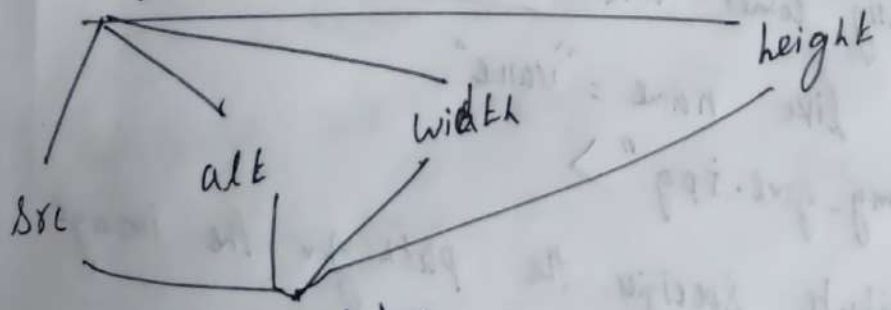
<h1> ..... <h6>

<a> → link tag

 This is a link

href → attribute

<img> → image tag



HTML Elements → Attributes :

 This is my link

- Element which starts from a starting tag to the ending tag.
- The HTML element has no content are called empty elements [eg] →
.
- HTML is not Case Sensitive.

<P> != <p>

HTML Attributes

- All html elements can have Attributes
- It provides additional information about elements
- It is always special in Start tag.
- Attributes usually comes with name/value Pairs like name = "value"

<img src = "img-girl.jpg">

- Src Attribute Specifies the path for the image

Two ways to specify the src attribute.

1, Absolute URL : → links from a different webpage.

[Eg] src = "https://www.w3schools.com/images/img-girl.jpg".

2) Relative URL : → Describes the same file and on the same page. Links to the image that is hosted inside the website. If the URL is without slash it will be relative to the current page.

The width and height Attributes.

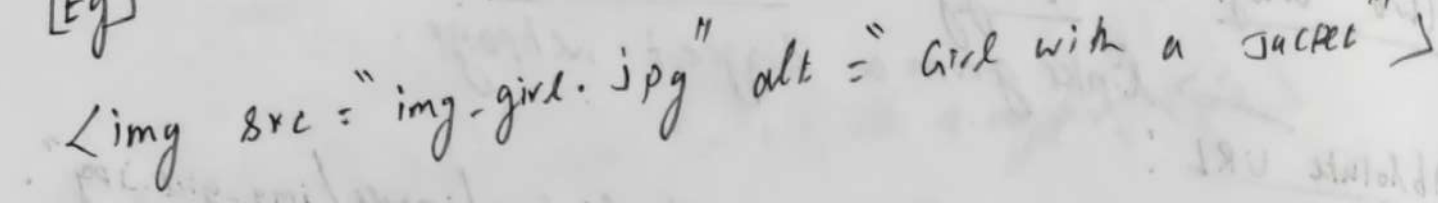
<img> tag should also contain width and height attributes that specify the width and height attribute in the images.

<img src = "img-girl.jpg" width = "500" height = "600" >

alt Attribute.

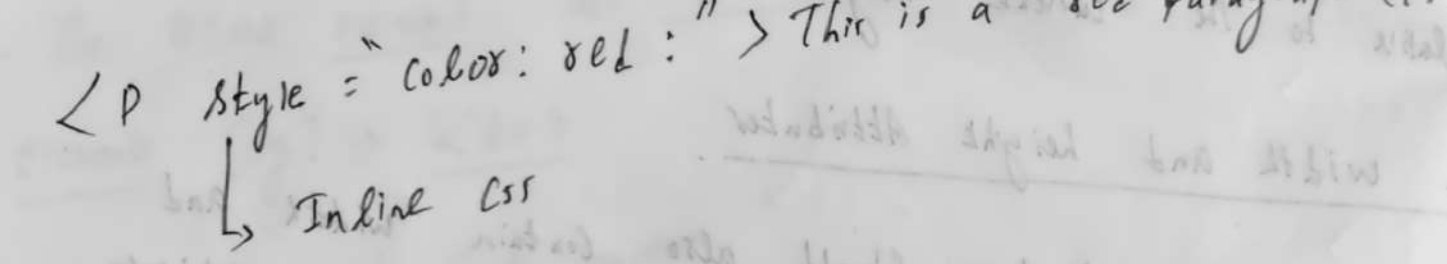
alt attribute is the alternative text for an image if the image can't display this alt will be displayed. if any error with the src attribute alt attribute will work and display.

[Eg]



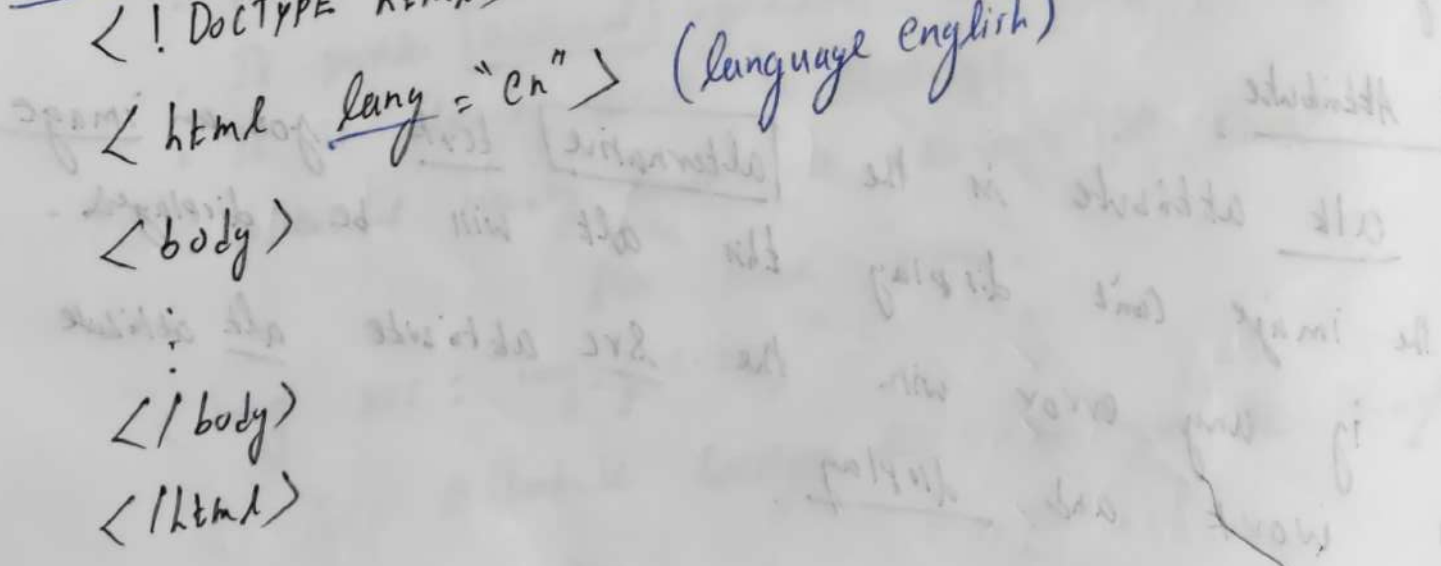
The Style Attribute

The Style attribute is used to add styles to an element such as color, font, size and more



The lang Attribute

You should always include the lang attribute inside the <html> tag, to declare the language of the webpage. This is meant to assist the search engine and browsers.



The title Attribute

The title attribute defines some extra information about an element

 This is a paragraph.

Always quote the attribute Values

Double and Single quoter can be used inside the Values of the attributes.

HTML Headings

. The html headings are defined with the <h1> to <h6> tags

<h1> defines the most important heading

<h6> defines the least important heading

. The html heading size can be a default. we can Change this using the Style attribute by using Css font-size property.

<h1 Style = "font-size: 60px:"> Heading 1 </h1>

HTML Paragraphs:

The HTML element defines a paragraph

HTML Horizontal Ruler:

<hr> tag often defines as a horizontal rule. and it is also used to separate content in the HTML page.

HTML Line Breaks:

 defines the line break

This is
 a paragraph
 with line breaks.

HTML <pre> Element

. The HTML <pre> tag (element) defines the preformatted text.

. The text inside a <pre> element is displayed in a fixed-width font (usually Courier) and it preserves both spaces and line breaks.

Elements & Attribute Conclusion.

 Paragraph
<hr> horizontal row

 break
<pre> pre-formatted text

HTML Styler Attribute.

Syntax.

<tagname style = "property : value :" >

tagname (Element)

Attribute.

Background color.

The Background color defines the Bg-color of the particular html element.

<body style = "background-color : powderblue :" >

The whole page will resemble in a powderblue color.

 This is a paragraph
<h1> This is a heading </h1>

This mentioning in the body can resembles the particular & <h1> tag

</body>

The CSS Color property defines the text color for the HTML element.

[Eg] <h1 style = "color : blue :" > This is a heading </h1>
 This is a paragraph

Styles Fonts

The CSS - font-family property defines the font to be used for an HTML element.

<h1 style = "font-family : verdana" > This is a heading </h1>

 This is a paragraph

Text Size <-> font-size property.

<h1 style = "font-size : 300 % :" > This is a heading </h1>

 This is a paragraph

Text Alignment <-> style = "text-align : center ;"

<h1 style = "text-align : Center :" > Centered text heading </h1>

 Centered paragraph

Chapter Summary.

- Style
- background-Color
- Color
- font-family.
- font-size.
- text-align.

HTML Text formatting.

<b> Bold Text.

 Important Text.

 - italic text

 Emphasized text.

<mark> Marked text

<small> Smaller text

<del> - Deleted text.

<ins> - inserted text.

<sub> Subscript text

<sup> Superscript text

and <bold>.

 The content inside is typically displayed in bold.

 This text is important!

and Elements.

 The content inside typically displayed in italic.

 The content inside is Emphasized and it is also displayed in italic

<small> The small element defines the smaller text </small>

<mark> The mark element defines text that should be marked or highlighted.

 Do not forget to buy <mark> milk </mark> today.

< del > Element

< p > My favourate color is < del > blue < /del > red < /p >

O/p my favourate Color is ~~blue~~ red.

HTML < ins > Element.

< ins > insert element.

The text have been inserted into the document, the browser will usually underline inserted text.

< p > My favorate Colour is < del > blue < /del > < ins > red < /ins >
< /p >

O/p My favourate Colour is ~~blue~~ red.

. < Sub > Subscripted text < /Sub >

< Sub >
< p > This is a Subscripted < /Sub > text. < /p >

O/p This is a Subscripted text.

<Sub> → hairy the character below.

<Sup> → hairy the character above.

Quotation

<Q> where people live in harmony with nature </Q>

O/p "where people live in harmony with nature".

Abbrevation.

<P> The <abbr title = "World Health Organization"> WHO </abbr> War found in 1948. </P>

The WHO war found in 1948

HTML <address> Tag. (<address> Enter the full address </address>

. This tag defines the Contact information for the author / Owner of a document or an article.

. The contact information can be Physical address, Phone number, Social media handles, etc.

<bdo> Tag writer from Right to left. right to left </bdo>

Chapter Summary:

<abbr> → Abbreviation

<address> → Defines contact information for the author/own

<bdo> → defines text direction.

<cite> → defines the title of the work.

<q> → Defines the short inline quotation.

HTML Comments:

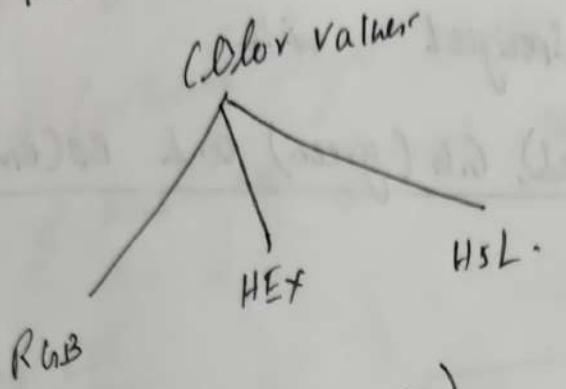
Syntax:-

<!-- write your Comment here -->

<!-- <p> This is a paragraph </p> This is a Comment which hides the paragraph -->

```
<!--
  <p> : </p>
  <br>
  <img> </img>
  -->
```

HTML has 3 Color Values



RGB → rgb (255, 99, 71).

Hex → #FF6347

HSL → (9, 100%, 64%).

RGB → Red, Green, blue. This means there are 256x256x256 Colors → 16777216 possible colors.

[Eg] rgb (255, 0, 0) displays as red

rgb (0, 0, 0)

Red Green Blue.

rgb (0, 255, 0) displays as green.

rgb (0, 0, 255) displays as blue.

rgb (255, 255, 255) displays as white.

~~No.~~

60 } Displays the
100 } shades of grey.
140
200
240

HTML HEX Colors

The hex colors are specified with

#RRGGBB, where the RR (red), GG (green) and BB (blue)

#ff0000 → Red

#00ff00 → Green.

#0000ff → Blue.

#000000 → Black.

#ffffff → White.

HTML HSL and HSLA Colors

HSL → Stands for hue, Saturation and lightness.

hsl (hue, Saturation, lightness).

0% is black.

100% is white.

hsla (hue, Saturation, lightness, alpha).

HTML Styles CSS

CSS → Cascading Style Sheets.

CSS is used to format the layout of a webpage.

Using CSS

- Inline - by using the style attribute.
- Internal - by using the <style> element in the <head> section
- External - by using the <link> element to link to the external CSS file.

Inline CSS

```
<h1 style = "color: blue;" > A Blue heading </h1>
```

Internal CSS

```
<!DOCTYPE html>
<html>
<head>
<style>
body { background-color: powderblue; }
h1   {color: blue; }
p    {color: red; }
</style>
</head>
<body></body>
</html>
```

External CSS.

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="Styles.css">
</head>
<body>
  <h1> This is a heading </h1>
</body>
</html>
```

. The file must contain the .css extension.

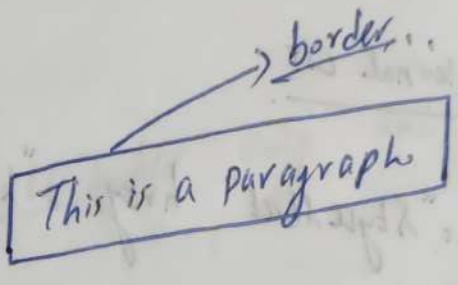
for format understanding.

```
<!DOCTYPE html>
<html>
  <head>
  <style>
    h1 {
      color: blue.
```

font-family : verdana ;
font-size : 300% ;

}
</style>
<body> </body>
</html>

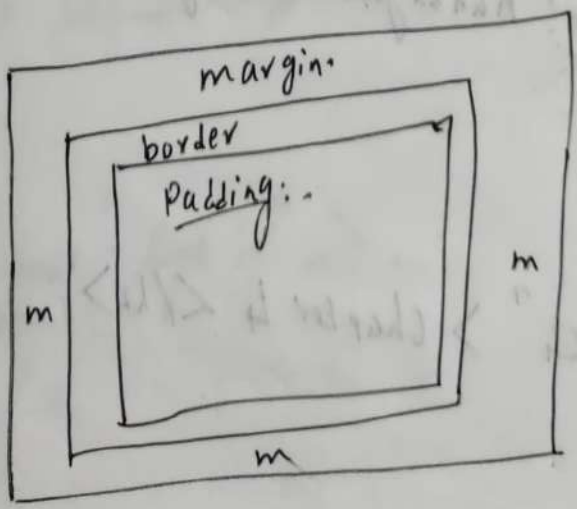
CSS border property.



O/P

<style>
p {
border : 2px solid powderblue ;
}
</style>

Padding <-> spacing.



rel attribute → Defines the relationship between the current document and the linked document.

Style element and related to CSS will goes inside the head.

link for External CSS.

<link rel="stylesheet" href="http://www.google.com">

.To make the image as a link, just put the image tag inside the a tag.

<img> <a>.

<img src="smiley.gif" alt="HTML Tutorial" style="width:42px; height:42px;">

.Send mail

ID attribute.

<h2 id="C4">Chapter 4</h2>

Id attribute declares the unique id for the HTML element.

Id attribute Syntax.

id = "value" (To use the id in the href).

href = "#value".

Image tag Container. the attributes only and does not Contains the closing tag.

Image takes two required attributes.

Image floating:

float is the CSS property. Can be used in CSS Styles:

Syntax.

<img Style = "float : left ; " >.

Syntax for background-image.

<p style="background-image : url('img-girl.jpg');">

↳ inline.

<style>

p {

background-image : url('img-girl.jpg');

}

</style>.

Background Cover.

- background-image : url( )
- background-repeat : no-repeat.
- background-attachment : fixed
- background-size. : cover.

- Picture Element <picture>.

↳ srcset (attribute)

<picture srcset="____" />

Favicon.

The favicon can be added for in the link.

Syntax.

<link rel = "icon" type = "image/x-icon" href = "/images/favicon.ico">

↳ Specific format.

HTML tables.

border: 1px solid black;

```
<table>
  <tr>
    <th> Company </th>
    <th> Contract </th>
  </tr>
  <tr>
    <td> Valeo </td>
    <td> 5 years </td>
  </tr>
</table>
```

Company	Contract
Valeo	5 years...

Table Cells.

<td></td> → table data.

<th></th> → table header.

<tr></tr> → table row.

Month	Savings
January	$100
February	$80
Sum	$180

Structure for table.

```
<table>
  <thead>
    <tr>
      <th> Month </th>
      <th> Savings </th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td> January </td>
      <td> $100 </td>
    </tr>
    <tr>
      <td> February </td>
      <td> $80 </td>
    </tr>
  </tbody>
  <tfooter>
    <tr>
      <td> Sum </td>
      <td> 180$ </td>
    </tr>
  </tfooter>.
```

Table Border.

Css border property on Table, th & td.

```
table, th, td {
    border : 1px solid black;
}
```

border-collapse : collapse ;

↳ To have a single type of border..

border-radius : 10px.

↳ border to get rounded corners..

border-style : dotted;

border-color : #96D4D4;.

HTML Table ColSpan & RowSpan.

The number of Columns to be Span.

<th ColSpan = "2" > Name </th>

<Col group> tag. → If we want to style the first 2 Colours we can use Colgroup..

HTML List.

Ordered list.

```
<ol>
  <li> </li>
  <li> </li>
</ol>
```

Unordered list.

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

Description list.

```
<dl>
  <dt> </dt>
  <dd> </dd>
  <dd> </dd>
</dl>
```

(or)

```
<dl>
  <dt> Coffee </dt>
  <dd> - black hot drink </dd>
  <dt> Milk </dt>
  <dd> - white cold drink </dd>
</dl>
```

Unordered list

```
<ul style = "list-style-type: disc;">
  <li> Coffee </li>
  <li> Tea </li>
  <li> Boost </li>
</ul>
```

disc → •
circle → ○
square → ■
No-list-marker →

Ordered list.

```
<ol>
    <li> c </li>
    <li> T </li>
    <li> M </li>
</ol>..
```

Ordered list have a Style type

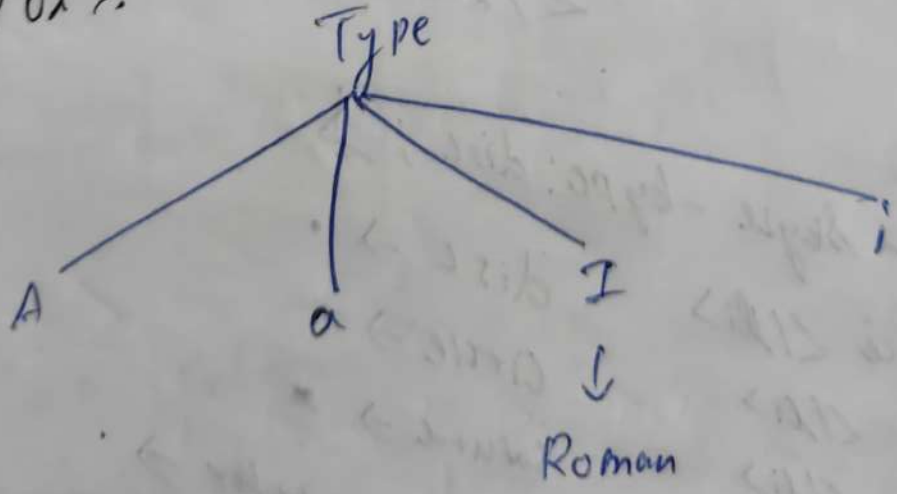
attribute..

```
<ol type = "A" >
    <li > coffee </li>
    <li> Tea </li>.
</ol>.
```

O/P

A Coffee

B Tea.



HTML Block and Inline Elements

<p> and

. Block level elements. Starts with a new line.

. There are many block level elements some are.

<p>, , <address>, <form>, <h1> - <h6>.

Inline element.

<Span>. Hello </Span>.

. It does not start with a new line.

 Element

↳ It is used to create the Container.
↳ The block of code need to be written inside the div.

<Span>

↳ This is an inline Container.

float property.

↳ is used for positioning and formatting Content.

And allows the elements to be horizontally.

display Property

block ;
inline-block ;

display : block ; → Vertical
inline-block ; → horrizontal.

Class attribute :

↳ (.) Symbol.

A Class can be added to many html elements :- more than one class
Different element can have the same class.

ID attribute.

↳ unique ID for html.
↳ (#) Symbol..

<form> Element

- used to create the form.

<label>
<input type = "text">.

Syntax.

<form>
<label for = "fname" > first name : </label>
<input type = "text" id = "fname" name = "fname" >
</form>..

<Input type = "text"
= "radio"
= "checkbox"
= "submit"
= "button..

<Select>
2 Option value = "Volvo" > Volvo </option>
<option value = "Bmw" > Bmw </option>.

form → action attribute, method
input → type, id, name, value
label → for.

<button type = "button" onclick = "alert ('Hello world!')">

July 11

CSS

property

h1 { color : blue → value ;

↳ CSS Element Selector.

CSS Id Selector.

#para1 {

text-align : center;

color : red;

}

Class Selector.

.center {

text-align : center;

color : red;

}

→ paragraph element with the center element

P.center {

text-align : center;

color : red

}

 This is a para

class1 class2

(*) Universal Selector.

Grouped the Selector.

h1, h2, p {

    ____ : ____

    ____ : ____

    ____ : ____

}

- Css
- BootStrap.
- media query (Responsiveness)

Css 29/5/25 10:50.

- Css -> Cascading Styles Sheet..

h1 {

↳ Selector.

    Color : blue; → property, → value.

    font-size : 12px;

}

Css Element Selector.

p {

    Color : red;

}

Css ID Selector

```
#para1 {
    text-align : center;
}
```

The Css Class Selector

```
.center {
    color : red;
}
```

Two Classes Reference

```
<p class = "center large" > This is a para </p>
```

center large

2 Classes...

```
* {
    text-align : center;
    color      : blue;
}
```

Group Selectors

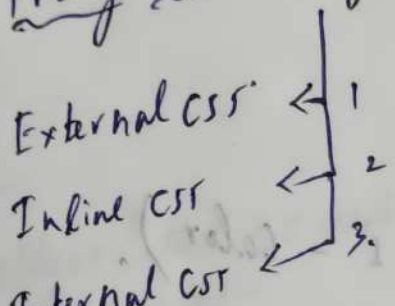
```
h1, h2, p {
    text-align : center;
}
```

Css (Adding)

- External.
- Internal
- Inline.

- The first priority is for inline CSS and the next is for internal CSS, before all of it External CSS has the first priority.

Priority wise Classification:



CSS Comments: /* This is a Comment */

HTML Comment: <!-- This is a Comment -->

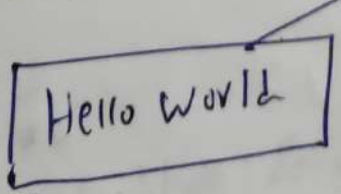
<h1 Style = "background-Color : Tomato;"> Hello world </h1>

↳ Changes the background of the block.

 Hello

↳ Changes the text color.

CSS border Color:-



→ Border Color in pixels

<h1 Style = "border : 2px Solid Tomato;"> Hello world </h1>

Css Color values

RGB, HEX, HSL, RGBA, HSLA.

rgb (255, 99, 27)

#ff6357

hsl (9, 100%, 64%)

Css Backgrounds (background-color).

```
p {
    background-color: blue;
    opacity: 0.3;
}
```

↳ makes the background to dull

(background-image)

[Eg]
```
body {
    background-image: url("paper.gif");
}
```

(background-position)

```
body {
    background-position: right top;
```

(background - attachment)

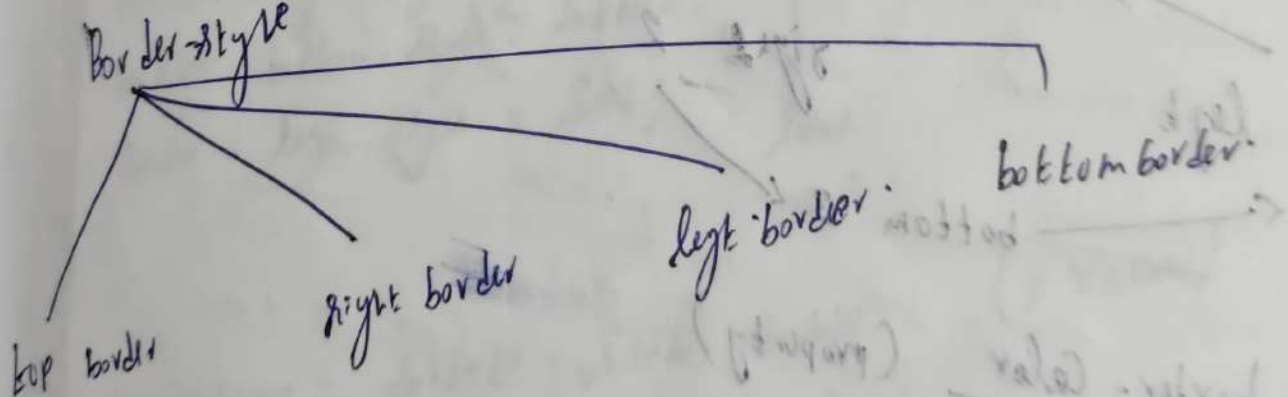
```
body {
    background-attachment : fixed;
                          : scroll.
}
```

CSS Borders

- Borders allow us the elements style width and colour of the element's border.

CSS Border-Style property

dotted, dashed, solid, double, groove, ridge ..... etc ....



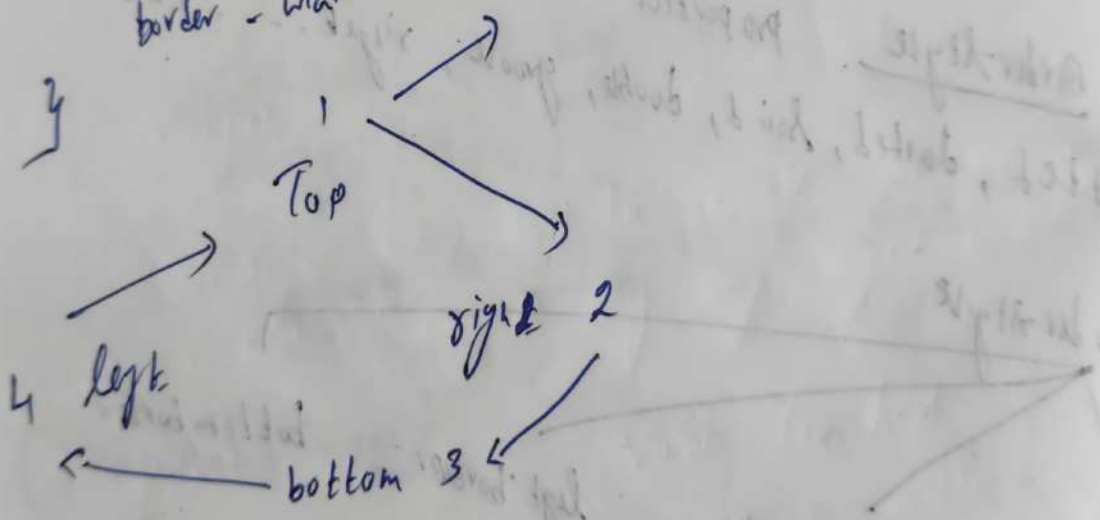
[Ex]

```
P. One {
    border-style : solid;
    border-width : 5px;
```

Specific side width

```
p. one {
    border-style : solid ;
    border-width : 5px 20px ;
}

p. three {
    border-style : solid ;
    border-width : 25px 10px 4px 35px ;
}
```



border-color (property)

```
p. one {
    border-style : solid ;
    border-color : red ;
}
```

It Can be in name → red
" Hex → #FF0000 "
RGB → "rgb(255, 0, 0)",
HSL → "HSL(0, 100%, 50%)";

P. One {
border-style : Solid;
border-color : red green blue yellow;

Css Border individual style.

. P {
Border-top-style : dotted;
Border-right-style : Solid;
Border-bottom-style : dotted;
Border-left-style : Solid;
}

P {
Border-Style : dotted Solid;
}

== Same result.

. borderStyle : dotted Solid double. (3 values)
↓ ↓ ↘
top border right & left bottom.

All properties in a single property

CSS Border — Shorthand property.

```
p {
    border : 5px solid red;
}
```

Rounded border.

↳ border-radius : 5px;

CSS Margins

- Margins are used to create the space around elements, outside of any defined borders.

Properties

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

Values Can Have

- auto
- length (px, pt, cm, etc)
- %
- inherit..

```
p {
    margin : 25px 75px 50px 100px ;
}
```
T R B L

- margin : 25px 50px 75px
  T R & L B
- margin : 25px ;
  ↳ all your margins
- margin : auto ;
  ↳ Center all the your sides.

Inherit it to use the value from the parent

Clear

```
div {
    border : 1px Solid red ;
    margin-left : 100px ;
}
p.ex1 {
    margin-left : inherit ;
}
```

(This margin left inherits the parent Clear and over the 100px left margin).

CSS Padding

To generate a space around the element Content, inside the defined border.

Padding.

- Padding-top
- Padding-right
- Padding-bottom
- Padding-left...

Value
- length (px, pt, cm, etc)
- auto
- inherit

width property

If the element has the Specific width, if the Padding added to the element will be added to the total width of the element.

box-sizing.

The box-sizing property keeps the width to 300px.

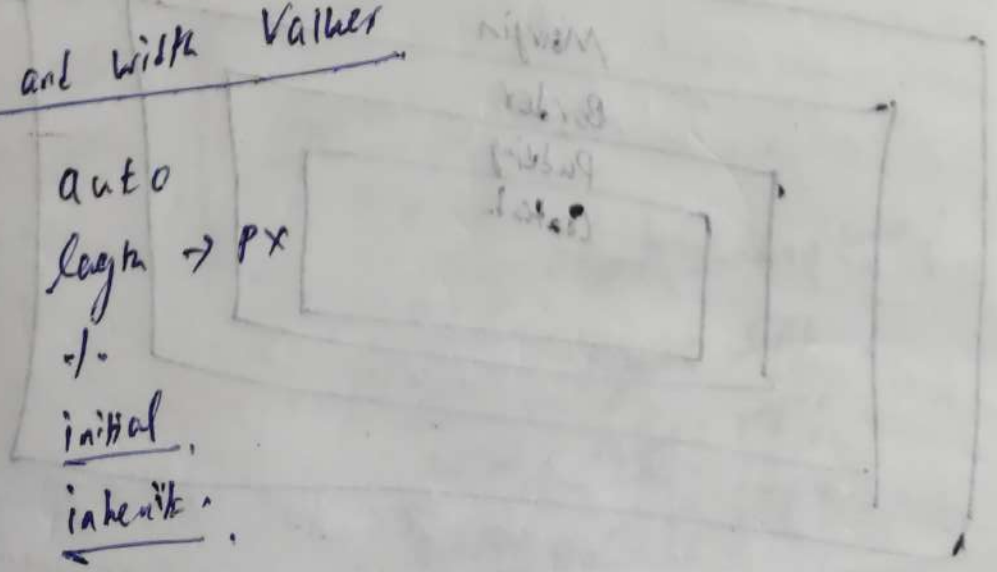
```
div {
    width: 300px;
    padding: 25px;
    box-sizing: border-box;
}
```

Css → Height, width and max-width.

- The height and width properties are used to set the hight and width of the Element.
- The max-width property defines the maximum width of the element.

Height and width Values

auto
length → px
-/.
initial.
inherit.

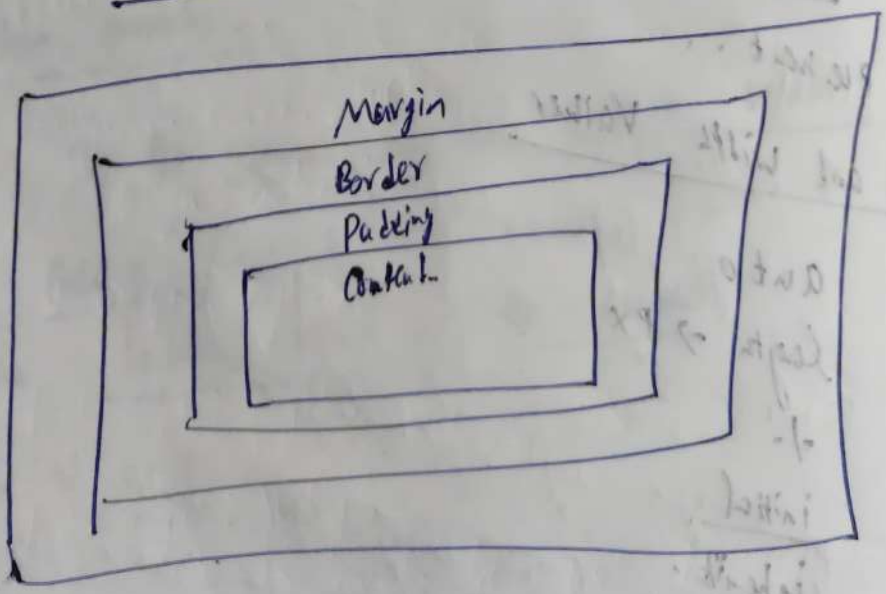


Note : The height and width properties do not include padding, border and margins, They set the height and width. inside the Padding, border and margin of the element.

All CSS Dimension properties ..

- height.
- max-height
- max-width
- min-height
- min-width
- width.

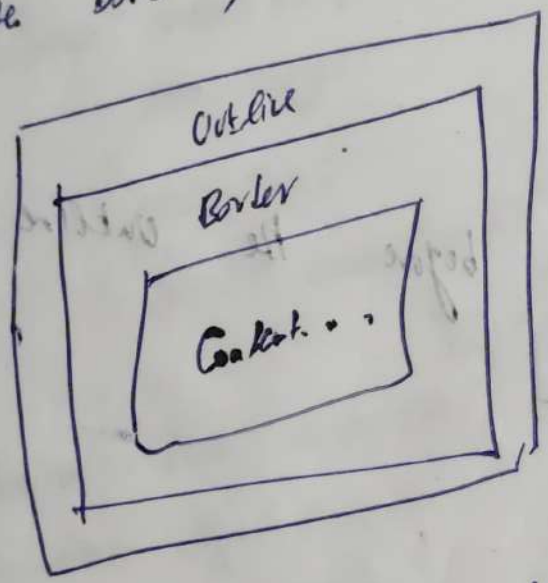
Box model



Total element width = width + left padding + right padding + left border + right border ..

CSS Outline:

The CSS outline is drawn around the elements outside the borders, to make the elements "stand out".



Outline-Style
outline-width
outline-color
outline-offset
Outline..

} Before the border the outline arrives.

Syntax

P.ex4{

border: 1px solid black;
Outline-style: solid;
Outline-color: red;
Outline-width: 4px;

}

Outline - Color : Red → rgb (255, 0, 0);
↳ property. hsl (0, 100%, 50%)

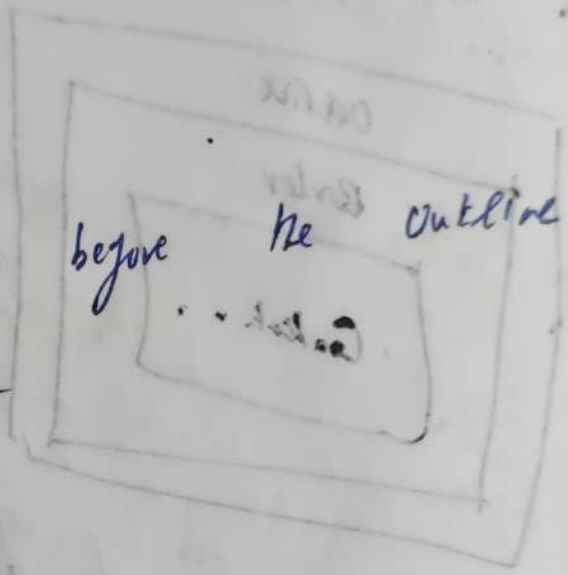
Outline - Style : Solid.
↳ property

Outline - width.

Outline - Style.

Outline - Color

Outline - Offset → Comes before the outline.



Css Text

. Color : blue ;

. background - Color : black.

. text - align : center ;

. text - align : left ;

. text - align : right ;

. text - align : justify ;
↳ makes the text to align in a proper way in a proper format.

. text-align-last : right ;

. direction : rtl ;
↳ Right to left text align direction.

. Vertical-align : text-bottom;
: sub;
: super;

. text-decoration : overline ;
: line-through;
: underline;
: overline underline;
:

. text-decoration-line : underline;
. text-decoration-style : solid;
. text-transform : uppercase ;
. text-transform : lowercase ;
. text-transform : capitalize ;
. text-shadow : 2px 2px red;

CSS Fonts :

. Serif
. Sans-Serif
. Monospace
. Cursive
. fantasy.

- font-family : _____ ;

- font-style : normal ;

- font-style : italic ;

- font-style : oblique ;

- font-weight : normal ;

- font-weight : bold ;

- font-variant : normal ;

: small-caps ;

- font-size : 40px ;

: 2.5em

1 em = Current font size.

Default text size 16 PX

VW → viewport width.

<h1 style = "font-size : 6vw"> Hello </h1>

CSS Icon.

Font awesome icon

The icon should be added inside head tag within a script tag.. tag.

<link src = " - - - - - " crossorigin " " ></script>.

.

CSS links (Styling)

. a : link
. a : visited
. a : hover.
. a : active.

} ⇒ Properties and Values...

. text - decoration : none.
. text - decoration : underline.
. background - color : yellow;

tr : nth - child (even) { background color : #F2F2F2; }

Responsive Table.

Block level html

→ Always starts on a new line.

<h1> <h6>
<header>
<footer>
<p>
<form>
<section>

→ Does not start on the new line.

Inline element

<a>
<img>

The position property

- Static
- relative
- fixed
- absolute
- stick y.

Position: relative ;

↳
Position: fixed
Position: absolute.
Position: sticky
↳ it acts when it scrolls.

- border-radius → Rounds the Corner.
- border-image: url (border.png) 40 stretch;

Css flexbox

- flex Container → the parent (container) element.
- Flex items → the items inside the Container

- flex box is mainly used for arranging items in row and Columns.

- It is a flexible responsive layout structure, without using float or positioning.

flex → properties

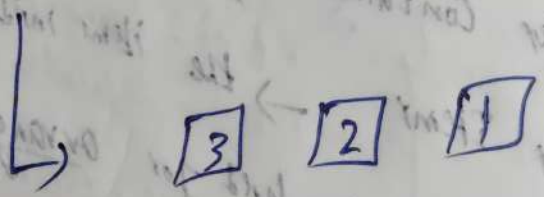
- display : flex ;
- flex-direction : column ;

↳ o/p

(•) .flex-container {

display : flex ;

flex-direction : row-reverse ;



flex-direction : column-reverse ;

(•) flex-wrap : nowrap ;

↳ Specifies the flex items will not wrap.

(·) .flex-Container {

display: flex;

flex-wrap: wrap;

}

(·) flex-flow : row wrap;

(·) The css Justify-Content property.

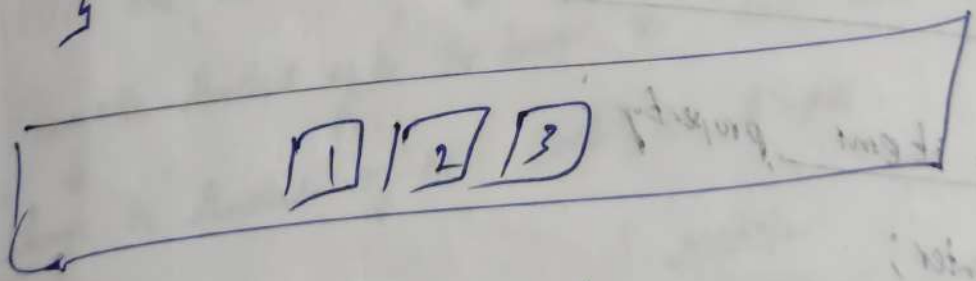
[Eg]

(·) .flex-Container {

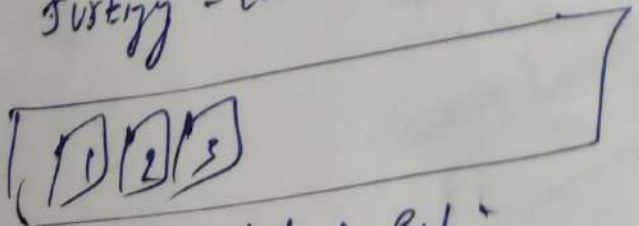
display: flex;

Justify-content: center;

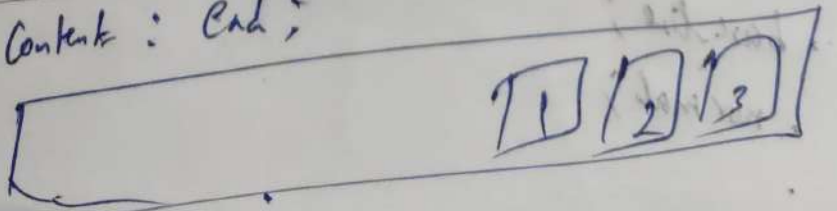
}



(·) Justify-Content : Start;



(·) Justify-Content : End;



. flex - Container {

display : flex ;

justify - content : space around ;

}

(-) justify - content : center

: flex - start ;

: flex - end ;

: space - around ;

: space - between ;

: space - evenly ;

The align - content property

. The align - items property :

. center ;

. flex - start ;

. flex - end ;

. stretch ;

. baseline ;

. normal ;

.

The CSS - align Content property.

↳ Similar to align-items;

- center,
- stretch;
- flex-start,
- flex-end,
- space-around;
- space-between;
- space-evenly;

main
header
section
section
section
footer ...

Changes.

- Click and redirect to a form
- Change the structure of the Top banner or col-md-4
- Change the structure of the card or simple.

main page

col

CSS.

1) Header

2) main
   - section
   - section
   - section
     - article

3) Footer..

```
.border{

}
div.border{

}
.border h1{

}
.border, .highlight{

}
```

Javascript

<p id="demo"> Javascript Can change HTML <p>

<button type="button" onclick="document.getElementById("demo").innerHTML="Hello Javascript!"> Click me </button>

Javascript → make web pages alive.

Javascript → Website manipulation.

Live Script ⟷ Javascript.

. Alert ("Hello")

<Script> </Script>.

Bro Code 12 hr

. Console.log ("Hello-world");

. document.getElementById("myH1").textContent = "Hello";

. let x = 10;

↳ Now x is a unique variable which should not be related for any other purpose.

let x = 123;

↳ here declaration and assigning to the number

Console.log(x);

↳ print 123

Console.log(`you are ${age} years old`);

Console.log(`The price is ${price}`);

let gpa = 2.1;

Console.log(`your gpa is ${gpa}`);

let age = 20.

Console.log(typeof gpa);

let firstName = "BRO-code";

Console.log(`your first name ${firstName}`);

let email = "nand123@gmail.com";

↳ anything can be added to the

String → Dynamic programming

Javascript Tutorial Doer. (Documentation).

- // alert ("Hello world");

↳ Comment in JavaScript.

- Console.log ("welcome nanda");
- Console.log (12345);
- Console.log (31.45);
- Console.log ([1, 2, 3, 4]);

↳ O/P → (4) [1, 2, 3, 4] → Json object.

- Console.log ({name: 'nanda', age: 25});

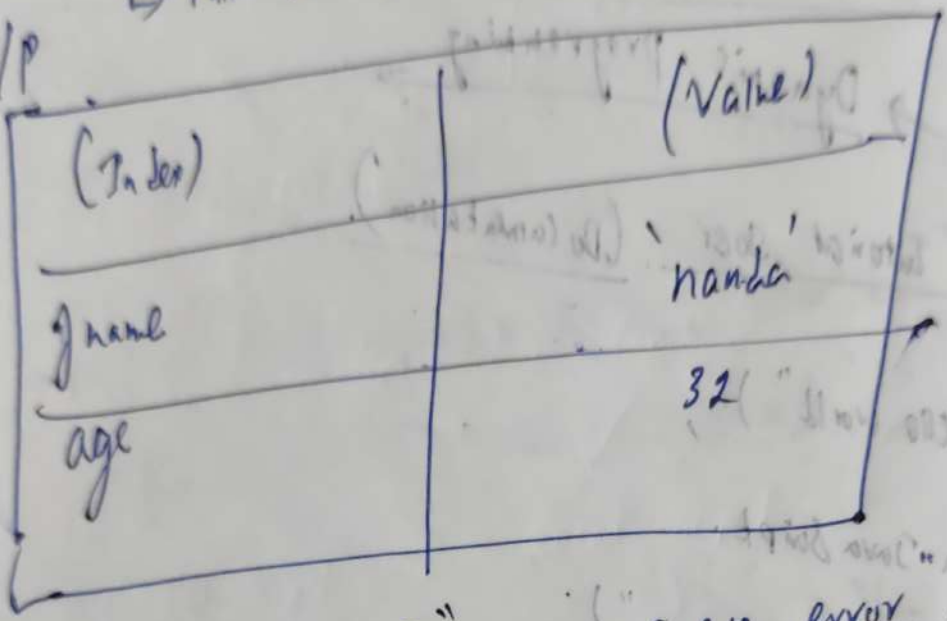
↓ ↓ ↓ ↓

Key pair Key pair

Console.table ({jname: nanda, age: 25})

↳ This will print in the table format.

o/p



(Index)	(Value)
jname	'nanda'
age	32

- Console.error ("Custom Console error");

↓

Creating the Customized error.

- Console.warn ("Custom Sample warning");

↓

Creating the Sample warning;

- Console.time ("Timer");
- Console.timeEnd ("Timer");

- Console.Clear()
  ↳ is used to clear the whole console.

Console.time("Timer");
↳ This is used to see how much time does the code runs in the execution time starts,

```
for (i=0; i<10; i++) {
    Console.log(i);
}
Console.timeEnd("TimerEnder");
```

O/P
1
2
3
4
5
6
7
8
9
10

Timer : 0.5229 ms.

Var let Const

→ Variable.

```
Var a = 25;
var b = 35;
Console.log (a+5);
```

1995 - 2015 → Regular Variable declaration.

(1) Scope.

Var → Global Scoped

let, Const → local Scoped.

[Eg] var.

```
if (true) {
    var msg: "I am good";
}
Console.log (msg);
```

O/p: I am good.

let & Const. → moreover works the same.

[Eg]

```
if (true) {
    let msg = "I am good";
    Const a = "nanda";
    Console.log (msg);
} Console.log (a);
```

→ This both will print bez local scoped.

Console.log (msg);
Console.log (a);
} This will get error..

(2) Variable redeclaration :

Redeclaration of same variable using let

let and Const:

Var

Eg) var a = 10;
Console.log(a);
var a = 30;
Console.log(a);

O/P 10
30.
} variable a can be redeclared..

let a = 10;
Const b = 20;
Console.log(a);
Console.log(b);
let a = 30;
Const b = 35;

Error
Identifier a and b has already declared..

(3) Value assignment

Var a = 10
Console.log(a);
a = 25;
Consol.log(a);

O/P [10, 25]

let a = 10
Console.log(a);
a = 25;
Consol.log(a);

O/P 10
25

Const a = 10;
Console.log(a);
a = 25;
Console.log(a);

O/P Type error
Assignment to a Constant variable..

const Student = { name : nanda , age : 18 };

↳ object declaration.

Console.table (Student);

Student.name = Gopal ;

Console.table (name);

O/P

→ [name : ram , age : 12]

age : 12

name : Gopal

Object.

(index)	(value)
name	Gopal
age	12

// Data types in JavaScript :-

String eg : "Nancia"

Number eg : 1.25, 25

Boolean eg : True, False

Null → keyword.

Undefined

Symbols ES6 → Established 2015

primitive data types.

Reference type eg objects

Array

Object Literals

Date and type.

[Eg]

```
var a = 2.42 ; → integer type
var name = "nand" ; → String.
var phone = null ; → object
var isMarried = True ; → boolean
var b ; → Undefined.
```

Symbols → The purpose of the Symbols is to create a random word generation.

```
Const a = Symbol();
Console.log();
```

o/p // Symbol();

Arrays and Objects

```
Var arr = [ 'C', 'C++', Java ];
Console.log(typeOf arr); // object.
```

```
Var Student = {
    'name' : "Nanda",
    'age' : 27,
};
// Object
Var d = new Date();
Console.log(d);
```

o/p Thu Jan 05 10:18:26 (IST).

(•) Type Conversion in Javascript.

- Any datatype can be converted through Type Conversion ... →

```
let a;
a = 10;
Console.log(a, typeof(a));   // number
a = (String)(10); String(a);
Console.log(a, typeof(a));   // String ...
```

(•) Type Conversion String to number

Even boolean can be converted ...

```
let a = true;
Console.log(a, typeof a);   // true boolean
a = Number(a);
Console.log(a, typeof a);   // 1 'number'
```

True → 1
false → 0

Array to Number Conversion.

↳ o/p "NaN"

↓

Not a Number.

ParseInt → will be Converted to Number.

ParseFloat → will be Converted to Float type.

(•) Type Coercion.

```
let a = "10";
let b = 30;
Console.log(a+b); // op → 1030

a = Number(a);
Console.log(a+b); // op → 40;
```

• Arithmetic operation :-

+, -, *, /

```
let a = 10;
let b = 20;
let c;
c = a+b ;
c = a-b ;
c = a*b ;
c = a/b ;
c = a%b ;
c = a**b ;
c = a++ ;
c = ++a ;
c = a-- ;
c = --a ;
```

• Assignment operator

```
let a=10;
a = a+5 ; // 15
  ↕
a += 5 ; // 20
a -= 5 ; // 15
a *= 5 ; // 90
a /= 5 ;
a %= 5 ;
```

// (Arithmatic assignment operation).

Mathematical function

Mathematical function ;

```
let c ;
c = Math.PI ;
```

// Math function :-

c = Math.round (4.9) // 4

. floor (5.3) // 5

. ceil (5.3) // 6

Check big num

. sqrt (90)

. abs (-85) // 85

. trunc (2.58) // 2

only call the Integer.

. pow (2, 2) // 4

c = Math.max (10, 20, 30) // 30

c = Math.min (10, 20, 30) // 10

c = Math.random ( )

c = Math.floor ((Math.random() * 50 + 1));

1 - 50

Random number generation

c = Math.sign(25);

Sign → positive → 1
negative → -1
0 → 0

Math.sin(45) → ✓

Comparison operators.

= (assignment operator)

== (both the values inside a and b are equal).

let a = 10;
let b = 20;
Console.log(a == b); // false.

a === b

Compares both data types are same.

Later Say, a = 10
b = "10" } a == b → True

a = 10
b = "11" a == b false.

↳ Checks up only value..

a === b // false.. (Data type)

Later Say. a = 10;
b = "10";

a = 10
b = 10 a === b // True Datatype and value are same.

a == b → Checks the Value only.

10 == 10

a === b → Checks the data type as well as the Value

10 === 10

! → Not equal → Checks the Value;

a = 10;
b = "10";
a != b ⇒ false

a = 10;
b = "25";
a != b ⇒ True.

Relational operators

let a = 10;
let b = 20;
console.log(a > b) false
a < b true
a >= b false
a <= b true.

Logical operator in JavaScript.

&& logical and
|| logical or
! logical not..

2 or 3 Conditions need to be checked at the same time, means we need to use logical operator

```
let mark = 25;
console.log ( mark >= 25 && mark <= 30);
  └ true.
let a = true;
console.log (!a);
  └ false.
let a = 10;
console.log (a == 1 || a == 5);
  └ false
```

Strictly equality or Identity operator.

let a = 10 ;

↳ Single equal → Assignment operator.

let b = '10' ;

Console.log (a === b) ; // True.

↳ Checks only value here 10

Console.log (a === b) ; // false ;

↳ Checks the value or even or data type..

Ternary operator :

[Ex] :-

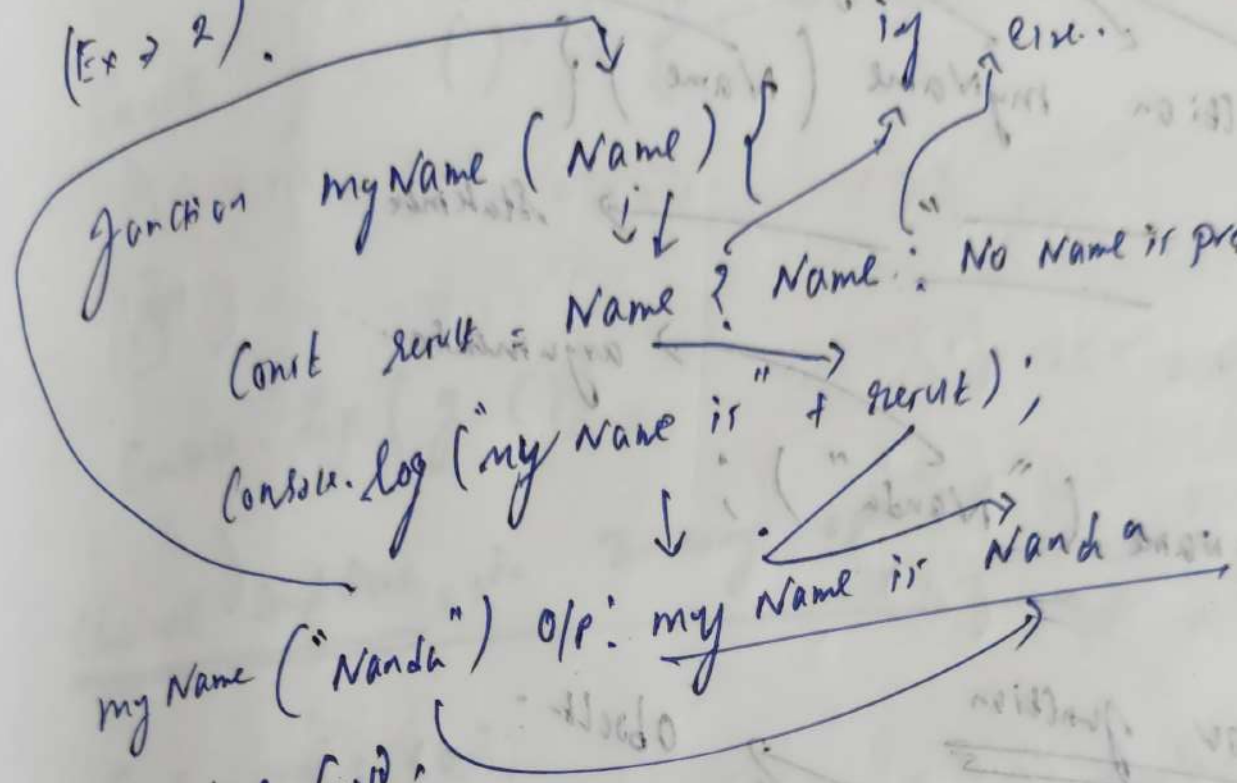
```
Const name = "nanda" ;
Const result = name ? "my name is " + name : "no name present" ;
Console.log (result) ;
```

o/p my name is nanda.

if

else

(Ex → 2).

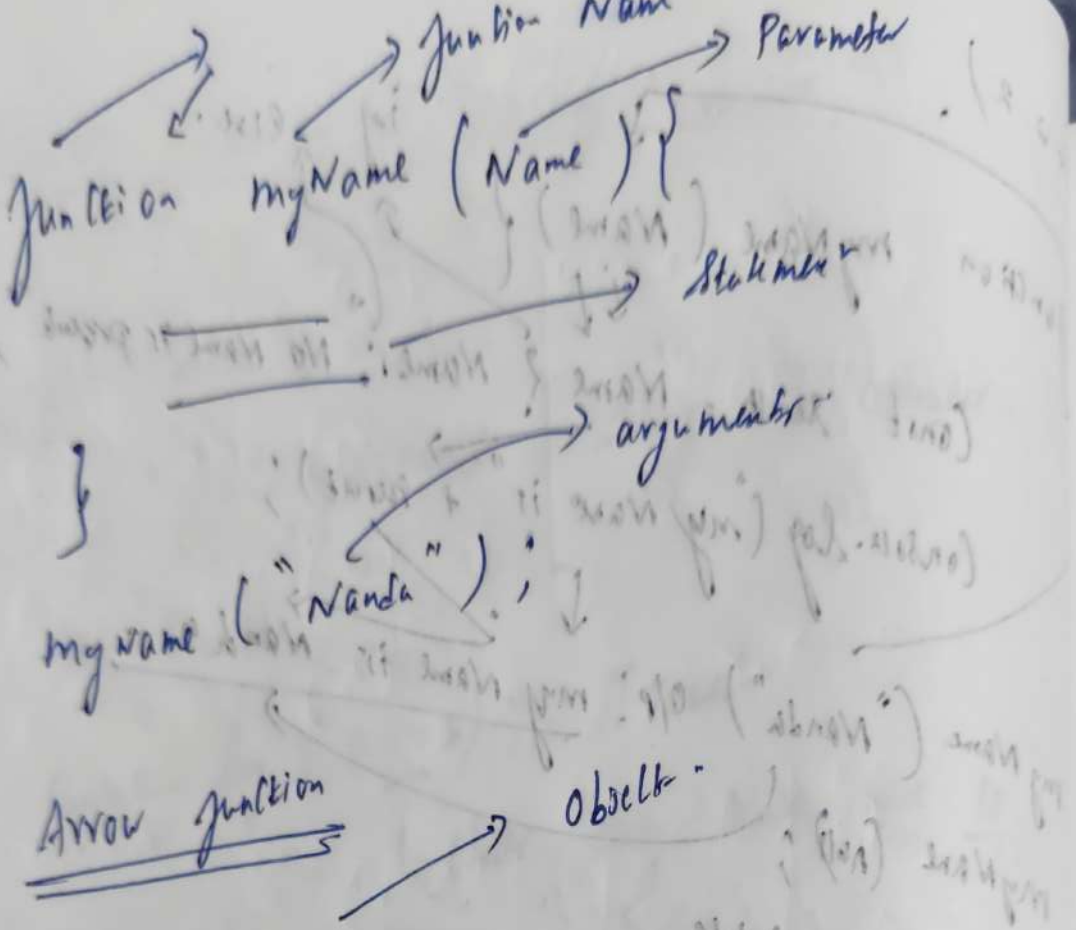


Object type variable

User = { name : nanda ,
age : 25 };

Console.log (user.name) // op → nanda.

- A function is a set of statements. Can be executed when it is called and it can be called (any) many times it can.




```
user = {
  'name' : "Nanda",
  'age' : 21
}
console.log(user.name)
```

anonymous function

// Nanda.

arrow fun.

```
const greetings = (user) => {
  const result = user.name ? user.name : "Nan
  return "Hello " + result ;
}
console.log(greetings(user));
```

// o/p Hello Nanda

Variable → anonymous function → syntax.

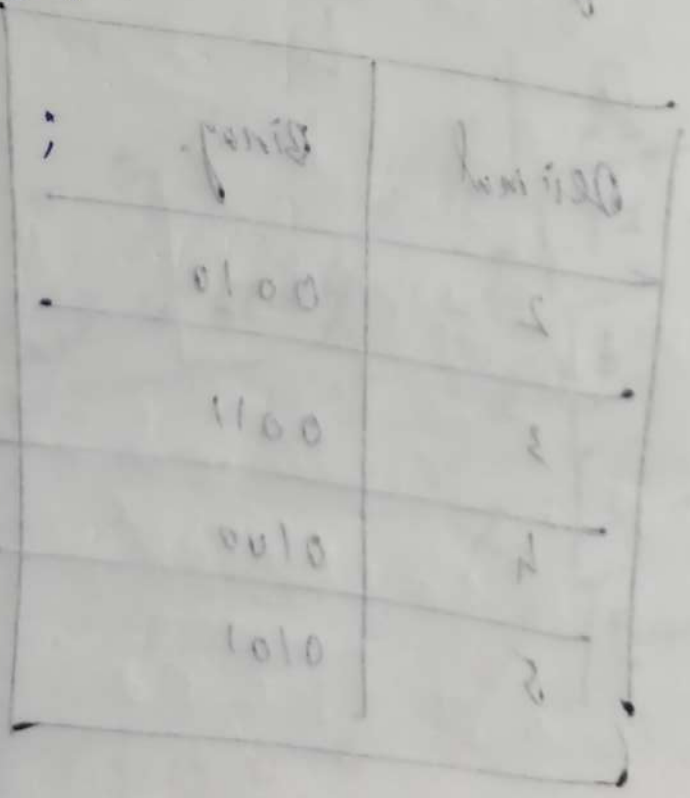
```
const g = () => {

};
console.log(g.());
```

Chained Conditions in Ternary operator

```
const avg = 90;
const grade = avg >= 90 ? "A grade" : avg >= 80 ?
    "B grade" : "C grade";
console.log(grade);
```

o/p → A grade ;



Bitwise Operator

Bitwise AND (&) Bitwise AND assignment (&=)

Bitwise OR (|) OR " (|=)

Bitwise NOT (~)

Bitwise XOR (^) XOR " (^=)

Bitwise &:

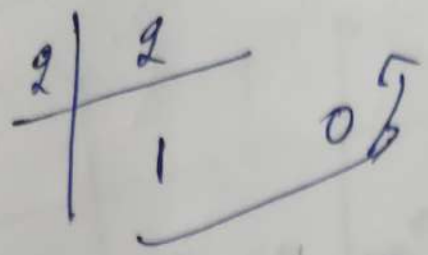
Left Shift (<<) (<<=)

Right Shift (>>) (>>=)

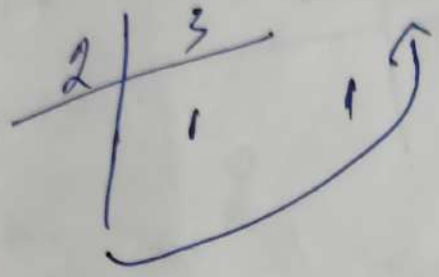
Unsigned right Shift (>>>) (>>>=)

Decimal	Binary
2	0010
3	0011
4	0100
5	0101

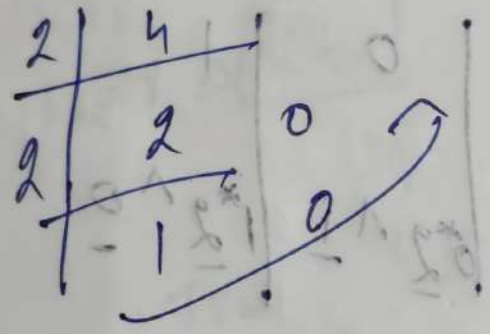
Decimal to binary



→ 0 0 1 0

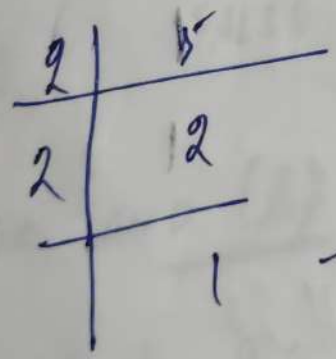


→ 0 0 1 1

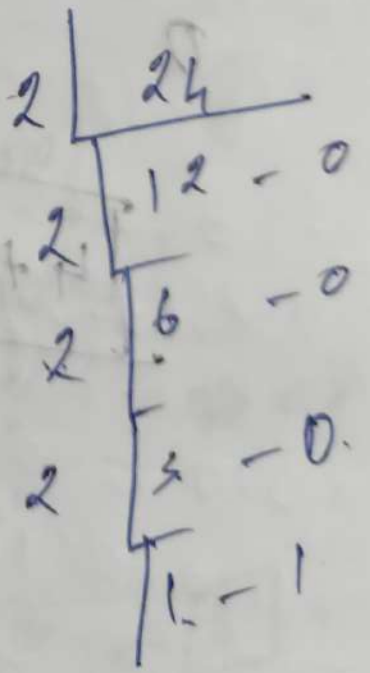


→ 0 1 0 0

→ 0 1 0 1



Decimal to Binay...



∴ 1 1 0 0 0

Binary to Decimal.

ex ⇒ 5

2 | 5
2 | 2 - 1
2 | 1 - 0

Ans 0 1 0 1

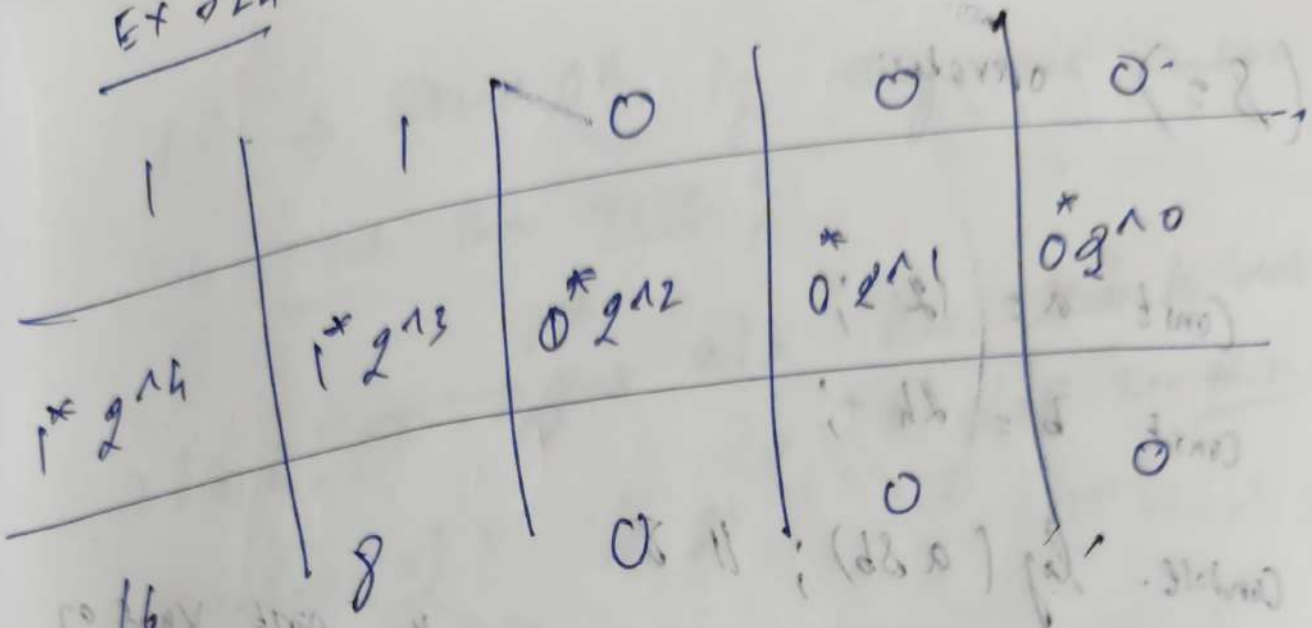
0 | 0 1 | 0 | 1

$0 * 2^3$ | $1 * 2^2$ | $0 * 2^1$ | $1 * 2^0$

0 | 4 | 0 | 1

0 + 4 + 0 + 1 ⇒ 5

Ex ⇒ 24



16 + 8 ⇒ 24

1 True. $2^5 <= 35$

0 FALSE $2^5 >= 35$

Bitwise AND (&)

Const a = 12 ; // (1100)

Const b = 24 // (11000)

Console.log(a & b) ; o/p ⇒ 8

0 0 0 0 1 1 0 0

0 0 0 1 1 0 0 0

0 0 0 1 1 0 0 0 ⇒ 8

(&=) operator:

```
Const a = 12;
Const b = 24;
Console.log(a &b); // 8
```

a &= b; → This checks the given value.

Console.log(a); // 8

a & b which is 8

then 8 is being stored at 'a'.

(|) → Bitwise OR |

```
a = 12;
b = 24;
Console.log(a | b); // 28
```

F O F O F O O F T | T F O O F

F O F O F O | T T | O F F O O F

F O F O F O | T | T F O O

(28)

| → Bitwise OR if any one condition is True then True.;

a |= b; → first finds a|b then stores the result in 'a' ⇒ 28 then prints.

Console.log(a); // 28

(~) Bit wise Not ~ [~a = -a - 1]

Short Cut ⇒ -25 - 1 ⇒ -26

a = 25;

Console.log(~a); // -26

(^) Bit wise XOR

a = 24 // 11000

b = 16 // 10000

01000

8 ← 1000 ⇒ 8

Same num 1 1 0

Diff num 1 0 1

(<<) leftshift:

a = 5

b = 2

Console.log (a << b);

// 20

→ 8 bit rep.

00000101 ← <<

00010100

5 → 0000 0101

<< 2 which says leftshift 2 so 2 0's will get (-) at first and added at last.

00010100

→ 20

a <<= b;

↳ 2o "Store in a"

(>>) Rightshift)

// >>

a = 5

b = 2

Consider (a >> b);

0 1 0 1 >>

→ (-)

0 0 0 1

Reduce two bits on the right side.

remaining ... if i add zero

Beyond it...

a >>= b;

Right shift by 2 and Store the value at

(a).

(??) Nullish Coalescing operator

```
const a = null ?? "No value";
console.log(a);
```

↳ if the value on the left is null then it print right value

o/p No value

else it will console print the left value

```
const b = 23 ?? 45;
console.log(b);
```

o/p => 23;

```
const user = { name: 'nanda' };
user.city ?? = 'Salem';
console.log(user);
console.log(user.city);
```

O/P

name : Nanda

city : Salem ,

Solution :

(++) Increment operator & (--) decrement :-

a = 1

a++ ⇒ a += 1

Console.log(a);

b = 5 ;

b-- → b -= b b = b - 1

a++ → a += 1 ⇒ a = a + 1

a-- → a -= 1 ⇒ a = a - 1

let a = 2 ;

Const b = a++ ;

Console.log(b, a) ;

O/P 2, 3

↳ Same as (a--)

let a = 5

Const b = ++a ;

Console.log(b, a);

O/P 6, 6

↳ Same for (++a)

If statement & else statement

```
if (condition) {

}
```

```
const age = prompt("Enter your age");
if (age >= 18 && age != null) {
    console.log("You are Eligible for voting");
}
else {
    console.log("NOT eligible");
}
```

Else if

```
if (Condition) {

} else if (Condi) {

} else {

}
```

Ex → 1

```
let num = 0;

if (num < 0) {
    console.log("your num is -ve", num);
} else if (num > 0) {
    console.log("+ve num", num);
} else {
    console.log("zero", num);
}
```

Nested if Statement.

(Syntax)

```
if (condition) {
    if (cond) {
        - -
    }
}
```

Switch Statement

Syntax

```
Switch (choice)
{
  Case Choice:
  ---
  break; → Compulsory
  Case Choice;
  ---
  break;
  default:
  ---
  break;
}
```

```
let num = 2
Switch (num)
{
  Case 1:
  Console.log(1);
  break;
  Case 2:
  Console.log(2);
  break;
  Default:
  Console.log("Invalid")
  break;
}
```

Combining Case Statement

```
let letter = 'a';

switch (letter) {
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
    case 'A':
    case 'E':
    case 'I':
    case 'O':
    case 'U':
        console.log(letter + " is a vowel");
        break;
    default:
        console.log("Nope");
        break;
}
```

//Looping statement

// while loop.

```
let i=1;
while (i<=10);
  console.log (i);
  i++;
}
```

// do while :

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

//for loop

```
for (let i=1; i<=10;
  console.log (i);
}
```

1
2
3
4
5
6
7
8
9
10

```
let arr = [];
for (let i=0; i<=100;
{
  arr.push (i);
}
console.log (arr);
```

Nested for loop :-

2D array

```
let numr = [ ];

for (let i=0; i<3; i++){
    nums.push([]);
    for (let j=0; j<3; j++){
        nums[i].push(j);
    }
}
console.table(numr);
```

0	1	2
0	1	2
0	1	2
0	1	2

For of Loop

```
for (let name of names) {
}
```

for (int i in n

in → Java

For in Loop

```
for (let prop in user) {
    console.log(prop);
}
```

Objects can be converted to the array using

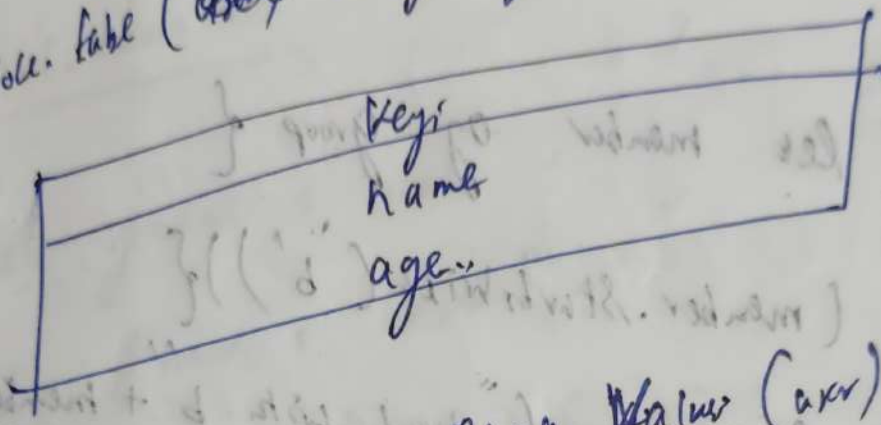
```
let user = {
    'name' : 'nanda',
    'age' : 16;
}
```

Object.keys(

Object.values(

let arry-keys = Object.keys(user);

console.table(arry-keys);



let arr_values = Object.values(user);

console.table(arry_values);

break and Continue statement

```
let groups = [
  [ 'a', 'b', 'aa' ],
  [ 'c', 'd', 'ac' ],
  [ 'e', 'f', 'g' ],
]
```

→ If label.

```
for (let group of groups) {
    inner:
    for (let member of group {
        if (member.startsWith('b')){
            console.log("found with b" + member);
            break inner;
        }
    }
}
```

When it found the first b and moves to upper loop

it breaks

Javascript String methods.

Concatenation :

```
let fname = "nanda";
let lname = "Gopal";
let ser = fname + lname;
Console.log (ser);
```

O/P nandagopal.

Concat

```
// Concat.
C = fname.Concat (" ", lname);
```

O/P nanda gopal.

(+=) ⟶ append.

// append

```
C = "Tutor";
C += "Nanda";
Console.log (C)
// Tutor Nanda.
```

// Escaping.

```
let s = 'She can\'t run';
```

↓

Escaping.

// Length. `let girlt_name = "nanda";`

```
let c = girlt_name.length;
console.log(c); => 5
```

// Upper Case.

```
let c = girlt_name.toUpperCase(); NANDA
```

||

```
let c =          toLowerCase(); nanda
```

// indexOf :-

```
c = first_name.indexOf('d');
console.log("indexOf d: " + c);
```

↳ ans → 3

0 1 2 3
n a n d a

// charAt

```
c = first_name.charAt(1);
console.log("charAt 1 : " + c);
```

a

// charCodeAt :-
↳ Character ASCII Value will be printed.

// Substr

```
c = first_name.substr(0, 3);
console.log(c);
```

n a n d

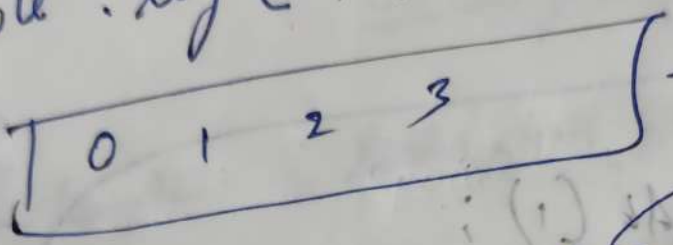
// Substring function

let text = "0 1 2 3 4 5 6 7 8 9 0";

c = text.substring (0, 4);

console.log (c);

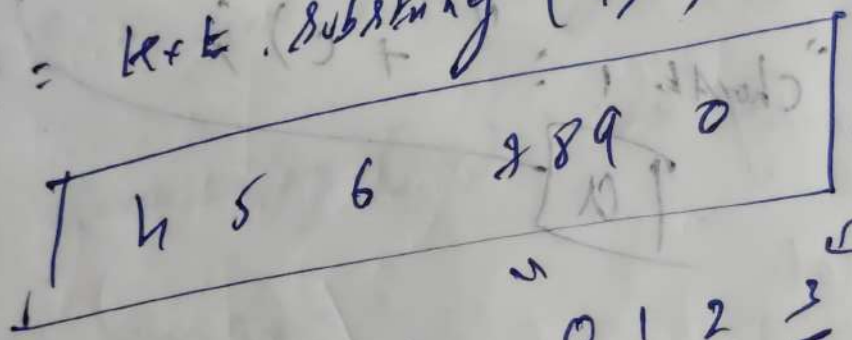
start, end.



→ o/p.

c = text.substring (4);

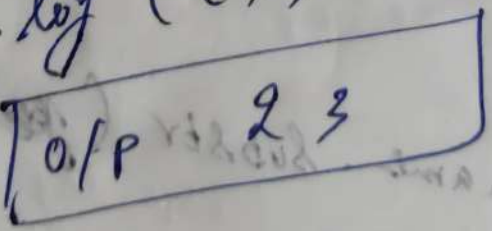
after 3



// slice ( ) text = 0 1 2 3 4 5 6 7 8 9

c = text.slice (2, 4);

console.log (c);



slice (-3).

↳ 8 9 0 o/p

// Split in JS.

let a = "Tutor Joer Computer Edu";

c = a.split(" ");

Console.log(c);

Tutor, Joer, Computer, Education

O/P

// replace in JS;

a = "I am from Chennai";

Console.log( );

c = a.replace('Chennai', 'Salem');

I am from Salem

O/P

// includes in JS

↳ Checks whether the data present in an array or not

return → True or false.

// Trim:

↳ reduces the space;

// padStart & padEnd.

// Long literal strings

↳ \

// fromCharCode ( );

Console.log (String.fromCharCode (65, 66, 67, 68));

↳ ABCD → O/P

- document.body.innerHTML = output;

- ${name}
  - This will be considered as the variable.

Array → array.

let a = [10, 20, 30, 40];

Console.log(a);

o/p → 10, 20, 30, 40

Console.table(a);

Index	value
0	10
1	20
2	30
3	40

Console.log(a[1]);

(-) let b = new Array (10, 20, 30, 40);

(-) let c = new Array ("Joe", 30, true,
{ m1 : 100, m2 : 75 : m3 : 65 });

Console.table (c);

Array can have any type of values inside them.

25 Method in JS Array.

forEach	Push	from
map	Pop	isArray
Slice	Shift	filter
Splice	unshift	Flat
Concat	indexOf	reduce.
Sort	lastIndexOf	
fill	every	
includes	Some	
Join	find	
Reverse	findIndex	

For each

```
let number = [1, 2, 3, 4, 5, 6, 7, 8, 9];

number.forEach ((value, index) => {
    console.log ("index : " + index + " value : " + value);
});
```

O/P

0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10

Syntax

```
forEach (value, index, array);
```

map ()

Syntax :-

map (value, index, array);

→ optional.

```
Const number = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let sqrt = map ((value) => {
    return Math.sqrt (value); .toFixed(2);
});

console.log (sqrt);
```

↓

gives only 2 decimal number after the point.

... (three dots) → we need not repeat the code

↳ (... → Spread operator)

(·) Slice (start, end);

```
Const num = [1, 2, 3, 4, 5, 6, 7, 8, 9];
Console.log (num.slice (3));
```

O/P 5, 6, 7, 8, 9.

(·) Spiice (start, length, new elements).

↳ remove element in an array

```
Const n1 = [1, 2, 3, 4, 5];
let remove_ele = n1.Spiice (2);
Console.log (remove_ele);
```

↳ 4, 5

```
Console.log (n1)
```

↳ 1, 2, 3.

(-) n1. splice (2, 2);

Start Element — End element

o/p 1 2 5 6 7 8 9 10

(-) n1. splice (2, 2, 25, 36); → Inserting

o/p 1, 2, 25, 36, 5, 6, 7, 8, 9, 10

(or)

n1. splice (2, 2, [25, 36, 45]);

Even the Array element can be added inbetween them.

Concat ()

```
Const a = [10, 20, 30];
Const b = [40, 50, 60];
Const c = [70, 80, 90];
let d = a.Concat(b, c, 'e', 'g', [14, 15, 16]);
Console.log(d);
```

10, 20, 30, 40, 50, 60, 70, 80, 90, b, c, e, g, 14, 15, 16.

Sort (); 4.14.00

```
Const a = [10, 20, 30, 50, 2, 42, 11];
Console.log(a.Sort());
Const namer = ["Kumar", "Aarun", "Joer", "Zara"]
Console.log()
Namer.Sort(); Console.log(namer)
```

O/P Aarun, Joer, Kumar, Zara.

Sorting the Number.

```
Const num = [10, 100, 25, 150, 45, 80, 90];
Console.log(num.sort());
```

10, 100, 150, 25, 45, 80, 90.
1 2 4 8 9

(Sorts based on the first number).

- Arrays can be properly sorted using a function

→ object one particular row

```
num.sort((a,b) => {
    return a-b;
});
```

b-a

↳ ascending sort ↓ Decending sort.

```
Console.log(num);
```

O/P → 10, 25, 45, 80, 90, 100, 150

// Fill method.

Fill (value, start, end);

Syntax

. The fill method is used to add the value to a certain parts.

[Eg].

let num = [1, 2, 3, 4, 5, 6, 7];

num.fill (20);

Console.log(num); → o/p → [20, 20, 20, 20, 20, 20, 20];

num.fill (20, 2, 5); → start. → end.

Console.log (num);

L→ o/p [1, 2, 20, 20, 20, 6, 7];

// Includes (value, start_index)

↳ optional parameter.

↳ It checks at the array whether the particular search value is present or not.

```
Const products = ["pens", "pencils", "Eraser", "Box"];

let Result = products.includes("pen");

Console.log(Result);
```

↳ O/P True.

- let Result = products.includes("pen", 2);

↳ Search index from 2nd Va

- Console.log(Result);

// array.join (separator),

↳ optional parameter..

↳ changes the array into a single line string..

[Eg] Const products = ["pen", "pencil", "Rubber"];

Console.log (products.join ('-'));

makes the array into a single line.

O/P - pen-pencil-Rubber..

separator

Console.log (products.join ( ));

O/P pen, pencil, Rubber.

Reverse ()

↳ Arrays and objects can be reversed easily, when there is length property is being applied

[Eg] Array Ex

```
Const num = [1, 2, 3, 4, 5, 6];
Console.log( num.reverse() );
```

O/P 6, 5, 4, 3, 2, 1

Obj Ex

```
Const obj = { 0: 10, 1: 20, 2: 30, 3: 40,
              length: 4 };
Array.prototype.reverse.call(obj);
Console.log(obj);
```

{ 1: 40, 2: 30, 3: 20, 4: 10 length: 4 }

push()

↳ add an element in an array...

```
let n = [1, 2, 3, 4];

n.push(44);
console.log(n);          → o/p 1, 2, 3, 4, 44.

console.log(n.push(55, 66));
```

↳ o/p [7]

↳ length will be returned.

```
console.log(n)
```

↳ o/p → [1, 2, 3, 4, 44, 55, 66]

// Merging two arrays by using push().

```
let fruits = ["apple", "mango"];
let vegee = ["Cucumber", "Tomato"];
```

→ Spread operator.

```
fruits.push(...vegee);
console.log(fruits);
```

o/p → [`apple`, `mango`, `Cucumber`, `Tomato`];

Pop ()

↳ Remove the element from the last index.

```
let fruits = ["apple", "banana", "mango"];
fruits.pop();
Console.log(fruits);
```

↳ o/p ["apple", "banana"];

∴ Even we can save this to a variable and can print which element is being popped

Shift ()

↳ Remove the element from the first index

```
let names = ["nanda", "Ram", "mohan"];
names.shift();
Console.log(names);
```

o/p → ['Ram', 'mohan'];

unshift ().

↳ Adding a new element to first.

```
let gnamer = ['Ria', 'mia', 'sia'];
gnamer.unshift('asal');
console.log(gnamer);
```

O/P → ['asal', 'Ria', 'mia', 'sia'];

---

indexOf ().

↳ Search the element index Particular position. in an array or well as String.

```
let gnamer = ['ria', 'mia', 'kia', 'pia'];
let i = gnamer.indexOf('pia');
Console.log(i);          // O/P → 3
Const Sub = "Nanda";
Console.log(Sub.indexOf('a', 2));
```

Search a from 2nd element.

O/P → 4

last Index of ()

↳ Some example. Searches from the last Index

Every and Some in JS } It can also be used in Objects.

```
let n = [2, 4, 8, 10, 11];

let result = n.every ((value) => {
    return value % 2 == 0;
});

console.log(result); // It is like an (&) operator
                     // every condition need to be true
```

↳ o/p false then it will return true.

```
let result2 = n.some ((value) => {
    return value % 2 == 0;
});

console.log(result2); // If any one condition
```

OR operator

Passes then every loop Pass or true. like (||

↳ o/p True

Primitive and Reference datatype

Primitive datatypes

- string,
- number,
- boolean,
- undefined,
- Symbol.

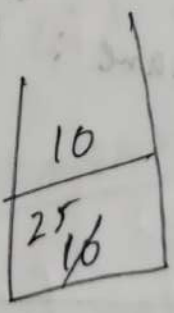
- let x;
  console.log (typeof x);
  O/P → undefined..

- let x = Symbol();
  console.log (typeof x);
  O/P → Symbol.

Primitive:

let a = 10;
let b = a;
a = 25;
console.log (a, b)

O/P
a = 25;
b = 10;



↳ It did not copy the memory so its time or lo

Reference Data type

In object ex. (=) Same for array

↳ It carries the memory address of the data

↓

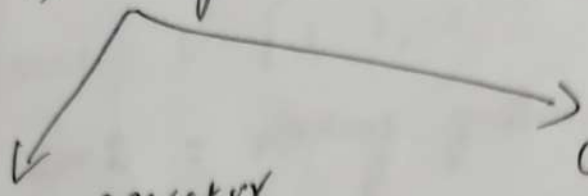
So it is called Reference data type.

```
let name1 = {
    name : "nanda",
    age : 23;
}
let name2 = name1;
console.log(name1);   {name : "nanda", age : 23}
console.log(name2);   {name : "nanda", age : 23}

name1.age = 45;
console.log(name1);   {name : "nanda", age : 45}
console.log(name2);   {name : "nanda", age : 45}
```

Object clone

↳ using



Spread operator

(or)

Shallow method

Object.assign().

ex

```
Const  obj1 = { a:1, b:2 };
Const  obj2 = { c:3, d:4 };
                {3,
Const obj3 = Object.assign(obj1, obj2);
                ^
Console.log(obj3); ↳ 1 method
```

o/p -> { a:1, b:2, c:3, d:4 }; →

2nd method Spread operator.

```
Const obj3 = { ...obj1 ; ...obj2 };
Console.log(obj3);
```

o/p { a:1, b:2, c:3, d:4 }.

Multiple ways to Clone an array.

// ... Spread operator

```
let arr1 = [1, 2, 3, 4];
let arr2 = [...arr1];
Console.log(arr2);
```

o/p → [1, 2, 3, 4].

// Cloned Array using Slice ().

```
let arr1 = [1, 2, 3, 4];
let arr2 = arr1.Slice();
Console.log(arr2);
```

o/p → [1, 2, 3, 4];

// Concat ()

```
let arr1 = [1, 2, 3, 4];
let arr2 = [].Concat(arr1);
Console.log(arr2);
```

// Array.from() :

```
let arr1 = [1, 2, 3];
let arr2 = Array.from(arr1);
Console.log(arr2);
```

→ Same O/P

// JSON.parse(), JSON.Stringify();

```
let arr1 = [1, 2, 3, 4];
let clone = JSON.parse(JSON.Stringify(arr1));
Console.log(clone);
```

//op Same

Const keyword working in array.

```
const arr1 = [1, 2, 3, 4];
arr1.push(5);
console.log(arr1);
```

o/p [1, 2, 3, 4, 5].

↓

So 5 can be pushed or edited but array

it cannot be changed or an whole array

Ex ⇒ arr1 = [5] → X

Arrays destructuring

```
let number = [10, 20, 30, 40, 50];
let [a, b, c, , d] = number;
console.log(a)   // 10
console.log(b)   // 20
console.log(c)   // 30
console.log(d)   // 50
```

Another 2 types of array destructuring:

```
let number = [10, 20, 30, 40, 50];
let [a, b, ...c] = number
console.log(c);
```

o/p → [30, 40, 50]

```
let nums = [10, 20]
let nested = [[1, 2], [3, 4]];
let [[a, b], [c, d]] = nested
console.log(b); // 3 o/p
```

Same for objects

left personal F.

Creating Objects in JS.

```
const person {
    name : "John",
    age : 30,
    job : "Developer"
};
console.log(person);
```

Objects literals of creation.

Object Constructor

```
const person = new Object();
person.name = "Nanda";
person.age = 19;
person.job = "Developer";
};
// Object.create(prototype, properties Object);
const personproto = {
    SayHello : function () {
        console.log("This is my name " + this.name);
    }
};
const person = Object.create(personproto);
Person.name = "Nanda";
Person.SayHello();
```

o/p -> This is my name Nanda..

by using new

```
Class Person {
    Constructor (name, age, job){
        this.name = name;
        this.age = age;
        this.job = job;
    }
}

Const person = new Person ("nanda", 19,
                           "Developer");

Console.log (person);
```

O/p object will be created.

Dot Notation and Bracket Notation

```
Const Person = {
    name : "nanda",
    age : 32,
};
Console.log (person.name);      // nanda.
                  Dot..
person.age = 40;                // 40..
Console.log (person.age);
```

// Bracket Notation..

```
Const person = {
    "first name" : "Tutor",
    "Last name" : "Joes",
    age : 24,
    job : "devops",
}
```

```
Console.log(user["first name"]);
```
↳ Tutor.

user["first name"] =

```
let key = 'first name';
Console.log(user[key]);
```
↳ O/P Tutor.

Iterating through object literals

```
const person = {
    name: "Nanda",
    age: 23,
    job: "Developer"
};
```

```
const keys = Object.keys(person);
console.log(keys);
keys.forEach(key => {
    console.log(`${key}: ${person[key]}`);
});
// same as values will be copied...
```

Objects inside array in JS

```
const user = [
    { name: "nanda", age: 20, email: "abc@gmail.com" },
    { name: "Jurgen", age: 30, email: "bca@gmail.com" },
    { name: "Ram", age: 40, email: "cba@gmail.com" },
];
```

```
for (let user of users){
    console.log(user.name);
}

const oldUser = users.filter(user => user.age > 30);
console.log(oldUser);
```

Function

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

O/P 30,

```
// function without argument

function sum(){
    let total = 0;
    for(let i=0; i< arguments.length; i++){
        total += arguments[i];
    } return total;
}
console.log(sum(12,12));
```

Arbitary Arguments

arguments

(i)

O/P

24

Function with inbuit operator and function with Spread operator.

Eg

```
function sum ( ... args ) {
    int t = 0;
    for ( let i=0; i< args.length; i++ ) {
        t += args [i];
    }
    return t;
}
console.log ( sum ( 1,2,3,4,5 ) );
```

Spread operator → ...args
Variable. → args

O/P → 15.

↓ Using spread operator here Values are being carried upon.

// function as an Expression..

```
const add = function (a,b) {
    return a+b;
}
console.log ( add (1,2) );
```

O/p	3

Arrow function

Syntax & parameter → let number = (a,b) => {

```
    return a+b;
}
console.log ( number (10,20) );
```

O/p	30

//

Arrow function by using map

```
let numbers = [1, 2, 3, 4, 5];
let doublenumbers = numbers.map(number => number*2);
Console.log(doublenumbers);
```

O/p [2, 4, 6, 8, 10,]

Using arrow function with filter ()

Ex

```
let fruits = ['applee', 'banana', 'mango', 'kiwi'];
let filteredfruits = fruits.filter(fruit => fruit.length >5);
Console.log(filteredfruits);
```

O/p => [applee, banana]

fruit => var

main variable assign.

Using reduce ()

```
let numbers = [1, 2, 3, 4, 5];
let result = numbers.reduce((sum, num) => sum+num, 0);
Console.log(result).
```

one by one number.

fruit. 2 var total will be added

O/p 15

final total value will be in sum.

Using arrow function create Closure.

```
let CounterCreater = () => {
    let count = 0;
    return () => {
        count ++
        return count;
    }
}
const counter = CounterCreater();
Console.log(Counter);
```

O/p 1

Block Scope

Block level Scope
- ~~Const~~
- let

Global Scope
- Var.

Const

```
if (true) {
  let BScope = "This is B Level Scope";
  console.log(BScope);
}
console.log(BScope); X
```

→ default Var will be assigned..

Function level Scope → Var

```
function myFunction() {
  functionVariable = "I am a Fvariable";
  console.log(functionVariable);
  function sum() {
    console.log(functionVariable);
  }
  sum();
}
myFunction();
```

O/P I am a Fvariable
I am a Fvariable

Diggreed beta — Rest parameter function one.

Spread operator.

// Rest parameter → used in function.

```
function myFunction(first, second, ...third) {
  console.log(first);
  console.log(second);
  console.log(third);
}
myFunction(1, 2, 3, 4, 5, 6);
```

first, second, ...third
Rest Variable..

O/P 1
2
[3, 4, 5, 6]

Rest parameter will be stored in an array

Spread operator

```
let a = [1, 2, 3, 4];
let b = [...a, 5, 6];
console.log(b);
```

spread → variable.

o/p → [1, 2, 3, 4, 5, 6].

Parameter destructuring

(1) Object parameter destructuring.

```
function sayHello({name, age}) {
  console.log(`This is my name ${name}`);
  console.log(`This is my age ${age}`);
}
```

Console.log(sayHello

```
const person = {
  name: 'nanda',
  age: 26
};
console.log(sayHello(person));
```

o/p This is my name Nanda.
This is my age 26

(2)

```
let num = [1, 2, 3, 4];
function add([a, b, c, d]) {
  return a+b+c+d;
}
console.log(add(num));
```

setTimeOut() → It makes delay

setInterval() → It will call again and again.

JavaScript DOM

Document Object Model:

- Dom is not a JS language instead web API
- Dom → Can be use in any language.
- Through Dom API we can talk with the Browser & JS
- JS is a programming language that Browser recognize

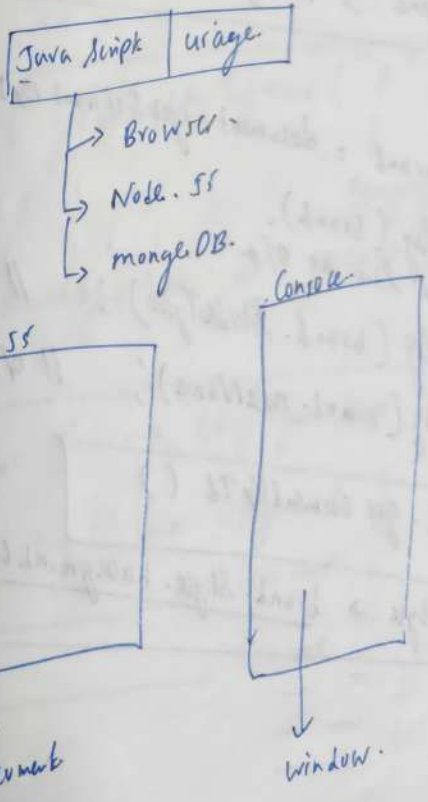
Dom — Dom API — JS engine

- To create an interactive website we use Dom

To access the Dom we use Dom API

What can be done through Dom.

- Create a new dynamic HTML element
- Change attributes values dynamically..
- We can add Event listener..
- Remove HTML Element.
- Change CSS Styles by applying to element



- Everything in JavaScript is an Object.

Altering a DOM Element.

All tags are called as nodes here

→ element → node → [1]

<h3 id="Brand"> Altering the DOM </h3>.

JS. Script.JS

```
let brand = document.getElementById("Brand");
Console.log(brand).
→ whole line as O/P.
Console.log(brand.NodeType);   // O/P → 1
Console.log(brand.nodeName);   // O/P → h3
```

(1) document.getElementById()

! To Style → brand.Style.backgroundColor = "purple";

(2). getElementByClassName().
↳ HTML Collection.

let Stitle = document.getElementsByClassName("sub-title");

Stitle[0].Style.Color = "blue";
↓
mandatory for ClassName Selector

(3) document.getElementByTagName("p");
↓
Paragraph tag.

. every functionality same as getElementByClassName();

querySelector() → Only for loop can be used...

↳ will can pair. Tagname
Classname
IDname.

querySelectorAll() → Returns NodeList

↳ forEach loop can be used..

document.querySelectorAll("p")

↓

Element:

document.querySelectorAll(".brand");

↓

(.) Id.

document.querySelectorAll("#Sub-title");

↓

(#) Class.

NodeList [] ⟷ It is an Array type []

So forEach loop can be used.

If you can't understand search for web

↳ HTML Collection (BS)

↳ NodeList.

---

adding list to an UL.

```
const li = document.getElementById("li");
```

↓ Selecting the element.

```
let element = document.createElement("li");
element.innerHTML = "JavaScript";
li[0].parentElement.appendChild(element);
```

O/P

- C
- C++
- Java
- JavaScript

Summary

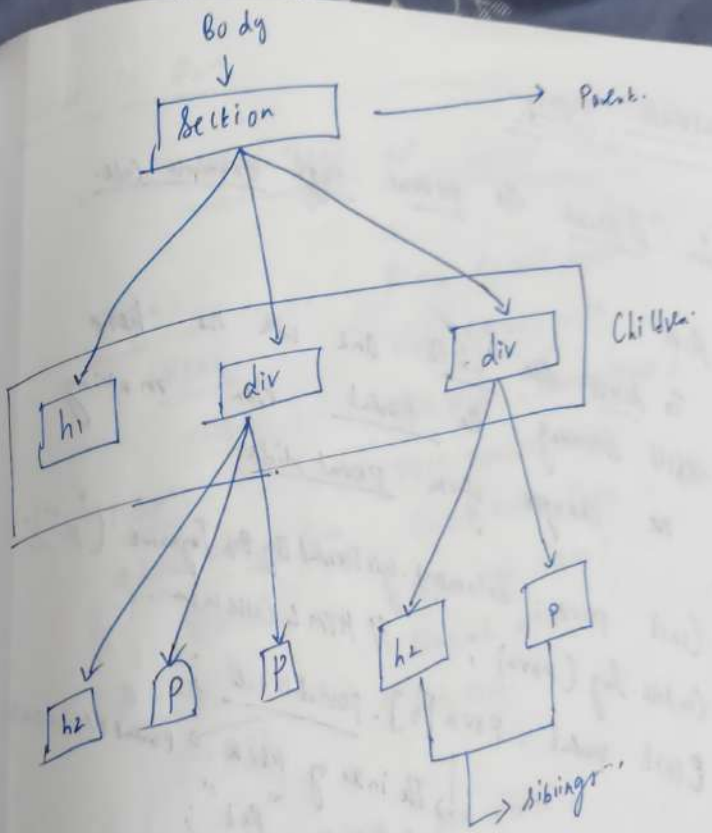
getElementById
getElementByTagName.
getElementByClassName.
querySelector &
querySelectorAll.

Traverse in Dom

Dom Tree

Ex

```
<html>
  <body>
    <section>
      <h1> This is a heading </h1>
      <div>
        <h2> Sample Title </h2>
        <p> This is a paragraph-1 </p>
        <p> This is a paragraph-2 </p>
      </div>
      <div>
        <h2> Sample Title </h2>
        <p> This is a paragraph-3 </p>
      </div>
    </section>
  </body>
</html>
```



ParentNode Property.

Ex. Assume the previous page example code.

Eg. Aim

To select the child and with the help of child selecting the parent then making up the changes from parent side.

Code.

```
Const para = document.getElementsByTagName("p");
Console.log(para);          // HTML Collection.
Const parent = para[0].parentNode;
                ↳ The index of para  → parentNode color,
parent.style.backgroundColor = "Red";
   ↳ By selecting the div the
```

by color will change

Node in Dom

. All HTML Tags are Element node

HTML → HTML Element Node.
Head → Head Element Node.
#text → Text Node (return, space)
Title. → Title Element Node.
#text Node in Dom → Text Node (Node in Dom)
#text → Text Node (text)
#comment → Comment Node
#text → Text Node.

Dom accessing the child and parent

Continuation of the Code.

firstChild
lastChild
firstElementChild
lastElementChild.

old code

// firstChild

```
const firstChild = parent.firstChild;
console.log(firstChild);
```

↳ o/p #text

// lastChild

```
const lastChild = parent.lastChild;
console.log(lastChild);
```

↳ o/p #text

// firstElementChild

```
const firstElementChild = parent.firstElementChild;
console.log(firstElementChild);
```

↳ o/p <h2> Sample text </h2>

```
const lastElementChild = parent.lastElementChild;
console.log(lastElementChild);
```

↳ o/p This is a paragraph!

- If there is no Element inside the firstChild or lastChild then it will return the text inside the element
- If there is no element inside the firstElementChild and lastElementChild then it will return the null.

---

children

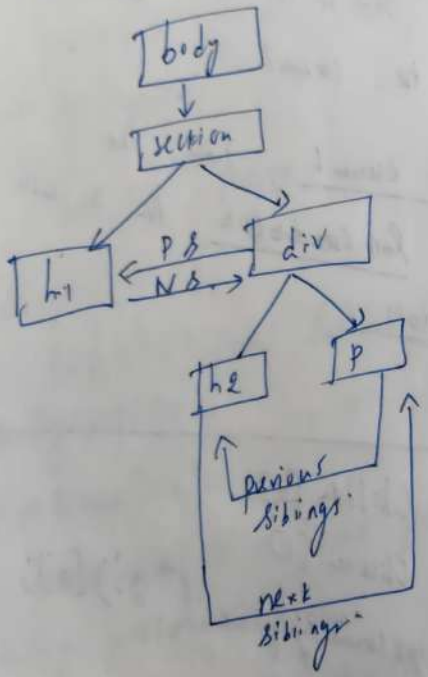
children[0]

children[1]

```
const div = document.getElementsByTagName("div")[0];
console.log(div);
console.log(div.children[0]);
```

↳ o/p firstchildren will be printed..

Siblings.



- previous Siblings
- previous Element Siblings
- next Sibling
- nextElementSibling.

```
Const h2 = document.getElementsByTagName("h2")[0];
console.log(h2.nextElementSibling);
```

o/p. This is paragraph

// Closest.

- The closest can only be called when using the querySelector
- which gives the closest value.

```
Const h1_tag = document.querySelector("h1");
Console.log(h1_tag);
Const Section = h1_tag.closest("section");
Console.log(section);
```

Topics Covering

createElement,
appendChild,
insertBefore,
removeChild,
remove,
cloneNode.

!

Example program.

. I need to add the paragraph element inside the body.

```
const let para = document.createElement("p");
para.innerHTML = "This is a <i> paragraph </i>";
para.style.backgroundColor = "blue";
let body = document.querySelector("body");
body.appendChild(para);
```

. Adding h1 before para

```
let h1 = document.createElement("h1");
h1.innerHTML = "This is a heading";
h1.style.color = "red";
body.insertBefore(h1, para);
```

Removing Btn remove() & removeChild()

```
let removeBtn = document.querySelectorAll(".btnremove");
removeBtn.forEach((btn)) => {
btn.addEventListner("click, function(){
const tr = this.parentNode.parentNode.
tr.remove()
~~const.log~~ {
let td = tr.childNodes[5];
console.log(td);
tr.removeChild(td).
```

- Style
- innerHTML
  ↳ In the dom the innerHTML can be used as the Tag inside the Text
- innerText.
  ↳ Normal Text can be added.
- CloneNode().
  ↳ is used to clone the same node and append it to the body element.

Syntax

```
CloneNode(True);
```
↳ If it wants to clone the innerText inside the element.
↳ The whole tag element will be cloned.

```
CloneNode(false);
```
↳ only the Tag element not the text inside the tag elements.

SetInterval()

Syntax.

```
SetInterval(SayHello, 1000)
```
SayHello ↓ Function
1000 ↓ Timer.

(Ex) Creating a clock

```
const ClockDiv = document.querySelector(".clock");

function clock() {
    const date = new Date();
    const time = date.getHours() + ":" + date.getMinutes()
    + ":" + date.getMilliseconds();
    ClockDiv.innerHTML = time;
}

SetInterval(clock, 1000);
```

DOM Part-5

Class List

```
classList.add();
classList.remove();
classList.toggle();
```

```
const btnAdd = document.querySelector("#btnAdd");
const box = document.querySelector(".box");
btnAdd.addEventListener("click", function () {
    box.classList.add("new-color");
});
```

Same like remove()

Same like toggle()

```
getAttribute();
setAttribute();
```

getAttribute();

↳ any Attribute Value can be get by using the getAttribute.

setAttribute();

input.setAttribute("type", "password");

↓ Type in attribute

↓ Change this to password.

.hasAttribute();

↳ Checker whether the attribute is present inside the element.

Returns true or false.

getAttributeNames();

↳ get what Attribute name was being present..

removeAttribute();

↳ Remove the attribute from the element..

JavaScript Dom Event Handlers

(1)

(a) `<button onClick="alert('welcome')">Click me</button>`

↓ listners

Inline event ~~handlers~~.

(2) Inline properties → using Script inside HTML.

```
<Script>
  const btn = document.querySelector("button");
  btn.onClick("click", function() {
    alert("welcome");
  });
</Script>
```

Event listners

↳ Using in External JS file.

With Event listner and a function..

Event listners

- click
- dblclick → double click..
- mousedown → one click it will change the event.
- mouse out → when the mouse pointer took out of the button mean it will change.
- mouseUP.
- mouse Over..

Keyboard events

- Key down
- Key press
- Key up

Functionalities happen when the keyboard key is being clicked.

`event.preventDefault();`

Form Event in Javascript

- Submit
- reset
- cleared
- blur
- focus
- change.
- blur
- focus
- input.

Touch events

- touchstart → mouse down event
- touchmove → mouse move
- touchend → mouse up
- touchcancel → Won't work in pc works at mobile.

Local Storage → Key - Value pair

- `localStorage.setItem(key, value);`
- `localStorage.getItem(key);`
- `localStorage.removeItem(key);`
- `localStorage.clear();`

(1) JSON.Stringify() Array -> String

↳ Convert Array of strings using JSON.Stringify() before storing.

(2) JSON.parse() String -> Array

. Edit inline edit

Promise in JavaScript

→ promise object.

If the problem is solved move it to the resolve parameter

else reject parameter

```
const promise = new promise ((resolve, reject) => {
  const sum = 1+1;
  if (sum == 2){
    resolve ("Success");
  }else {
    reject ("failure");
  }
```

Promise.then().catch()

↳ if the resolve parse it trigger the then().

If the reject parse it trigger the catch().

```
Promise.then ((msg) => {
  console.log ('msg');
})
.catch ((error) => {
  console.error ("error");
});
```

When a function that is Callback function is being called again and again while is nested inside the another Callback function is called as Callback hell

SetTimeout()

↳ normal function

```
SetTimeout ( ( ) => {
    Console.log ("Set time out normal function");
});
```

Set time out function with promise

Ex

```
function SetTimeOutpromise (duration) {
    return new promise ((resolve, reject) => {
        SetTimeOut (resolve, duration);
    });
SetTimeOutpromise (280). then (() => {
    Console.log ("Nanda");
});
```

O/p	Nanda

To prevent the callback hell we can use the promise function. And by using the then() we can make the callback again and again.

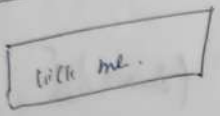
function

```
SetpromiseTimeOut (duration) {
    return new promise ((resolve, reject) => {
        SetTimeOut (resolve, duration);
    });
}
SetpromiseTimeOut (250). then (() => {
    Console.log ("Cool promise : 1");
    return SetpromiseTimeOut (250);
})
.then (() => {
    Console.log ("Cool promise : 2");
});
```

O/p	Cool promise : 1 Cool promise : 2..

Using the promise function in addEventListner.

<button> Click me </button>.




```
Const button = document.querySelector("button");

function EventPromise (element, method) {
    return new Promise ((resolve, result) => {
        element.addEventListener (method, resolve);
    });
}

EventPromise (button, "Click").then ((e) => {
    Console.log("button is clicked");
    Console.log (e);
})
```

Promise.all ( )

↳ returns an array [] when all the then() function is true.

. If one all function, Then the Catch() which is an error then all the array[] will return false.

. any ()

. allSettled.()

. race ()

AppScripts

- Logger.log() <=> Console.log();

Logger.log("Hello-world");

```
function learnBasics() {
  Var ss = SpreadSheetApp.openById("Put_Your_Id");
  Var sheet = ss.getSheetByName("AppScript - Basics");
  Sheet.getRange("D2").setValue("OMG!");
}
```

object → method → ID

Sheet Name.

Boilerplate.

getRange → Range

("D2") → D Column, 2 row.

setValue → value.

Sheet.getRange(9, 1).setValue("Hello");

9th row, 1st Column.

setValue..

Sheet.getRange(2, 2, 4, 6).setValue("Hello");

- 2nd row
- 2 Col To 4th row
- 6th Column.
- it will print the "Hello" value.

Getting the value and setting the value.

Let tempTxt = Sheet.getRange(13,1).getValue()

↳ The value was being get from the Sheet and it is being stored in the tempTxt Variable...

Sheet.getRange(16,5).setValue(tempTxt);

↓

Set the value to the particular range..

(2) Google Sheets - for loop, looping through cells, Variables, Comments.

// Get Current active Sheet
↓ Comment.

/* multi line
Comment */.

function learnBasics() {

Var app = SpreadSheet.OpenById (" ID ");

Var Sheet = app.getSheetByName (" sheet-name ");

~~Sheet~~.

let SomeValue = Sheet.getRange (8,1).getValue ();

Some value += 10;

Sheet.getRange (8,1).setValue (Some value);

O/P getting the value => 1

Setting the value -> 10 + 1 => 11.

For-loop.

for (let i=0; i<4; i++) {
  Logger.log ("Hello");
  Logger.log (i);
}

Same syntax like Java Script.

O/P
Hello
0.0
Hello
1.0
Hello
2.0
Hello
3.0.

for (let i = 8; i<12; i++) {
  let SomeValue = sheet.getRange (i,1).getValue ();
  SomeValue += 5.
  Sheet.getRange (i,1).setValue (SomeValue);
}

(3) Get and Set Value in Other Sheet.

Scenario:
1) getting the value from Sheet 2 and setting the value to Sheet 3.

Code.

```
function myFunction() {
  let app = SpreadSheetApp.OpenById("  Id  ");
  let targetSheet = app.getSheetByName("Sheet2");
  let Sheet3 = app.getSheetByName("Sheet3");
  let value = targetSheet.getRange(1,1).getValue();
  Sheet3.getRange(1,2).setValue(value);
}
```

(4) Clear Contents.

clear()
↳ will clear the data and formatting.

clearFormat()
↳ will only clear formatting.

clearContent()
↳ will clear data.

(5) Create CustomFunctions.

This is used to create a custom function and use this on the main Excel sheet.

(6) If-else.

. As usual if else statement followed by getting and setting the value accordingly.

a boilerplate

getValue ()

↳ working with a single cell

getValues ()

↳ working with many values.

.SetFormula ( );

Boilerplate.

```
function myFunction() {
  var ss = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var lr = ss.getLastRow();
  Console.log(lr);

//Sending an email code.
  for (let i=2; i<=lr; i++) {
  let CurrentEmail = ss.getRange(i,1).getValue();
  let CurrentTitle = ss.getRange(i,3).getValue();
MailApp.sendEmail(CurrentEmail, "Reminder: " + CurrentTitle +
                  "upcoming class", "Hello");
```

Syntax:

MailApp.sendEmail (recipient, subject, body);

↓ Current Email ID

↓ Any subject regarding mail.

↓ Actual message.

Ex email

Subject

body.

Appscript is object.

```
function learn() {
  var info = { name: "min",
               age: 32,
               eyeColor: "blue",
               gender: "female",
               favColors: ["green", "red", "blue"] };
  Logger.log(info["age"]);
}
```

o/p 32..

```
for (var key in info) {
  var val = info[key];
  Console.log(key);
  Console.log(val);
}
```

→ declares the key.

This declares the value from the key stored in val.

Web app Appscript

```
function doGet() {
  return HtmlService.createTemplateFromFile("index").evaluate();
  return HtmlService.createHtmlOutputFromFile("page");
}
```

→ when in this function above use to change.

→ need to be written in the HTML page.

google.script.run.userCreated();

↳ To run the userCreated function. which was written in the main Typescript file..

```
<?!= include("Styles"); ?>
```

↳ include the Styles page..

```
<?!= include("Script"); ?>
```

```
function include(filename) {
  return HtmlService.createHtmlOutputFromFile(filename).getContent();
}
```

is used to print Symbol ..

<?= ?>

Can write the JavaScript inside this Space.

<? var title = "hello"; ?>

To create a web app we need to use doGet().

Index.html (file) → Can write the JS Code.

<? var title = "hello"; ?>

 <?= title; ?>

(=) is used to view the JS Code on the Html Page.

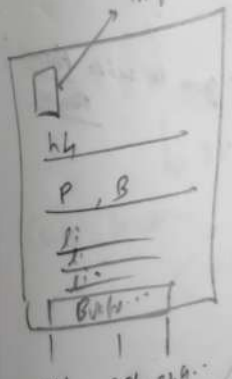
! → used when getting some kind of function is being used..

when it is not used means it will return any JavaScript Code or Html Code.

<? ?>

Text h3

h2



React JS 17/6/25..

- Created by facebook.
- React JS is a Javascript library for building front end application or UI
- Components are building blocks of any React app.

Adv

- Reusable Components
- Open Source
- Fast & efficient.

Command

```
npm create vite@latest.
```

- JSX → Javascript with XML
- JSX is a Javascript Syntax extension that allows you to write HTML in React
- JSX Converts HTML tags in React elements.

<> </>
↳ fragment.

Components..

Basic Component creation.

```
export const Header = () => {
    return (
        <div> Header </div>
    )
}
```

One Component should return only one child.

{ }
↳ JS expressions can be used inside the curly braces.

```
<p> 25 + 45 = {25 + 45} </p>
```
↓
70.

Inline CSS (optional)

```
<p className={customCss} style={{ fontSize: "20px",
fontStyle: "italic" }}> 25+45 = {25 + 45} </p>
```

Conditional Rendering

```
const isLoggedIn = false;
const greeting = isLoggedIn ? <p> welcome back! </p> :
                              <p> please login </p>;
```

JSX Summary

<> </> → fragment tags

{ } → write JS and HTML, CSS code inside for dynamic rendering.

Props <=> properties

Props
- What is props
- Default props

Works like a parameter when

The Parent Component passes the argument to the Child
- Parent Component: App.jsx
- Child: Student.jsx

React Hooks

UseState ()

Syntax

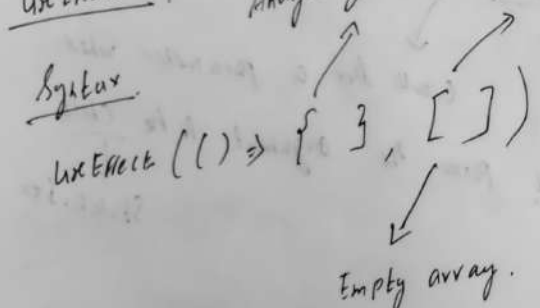
```
const [count, setCount] = useState( );
```

count → State variable
setCount → State function.

Whenever my State Value Changes my Component will rerender.

UseEffect.

Syntax



Clean up function in UseEffect.

[Eg]
```
function DemoComponent () {
    console.log ("Demo Component Mounted");

    useEffect (() => {
    Console.log ("useEffect in Demo Component");
    Const EachOne = setInterval (() => {
        Console.log ('Testing');
    }, 1000);

    return () => {
        clearInterval (EachOne);
        Console.log ("Clean up function");
    };
    }, []);

}.
```

Clean up function

---

useRef. To Stop the Component Rerender.

Component Rerender.

---

useContext :. Uncontrolled Input.

To Reduce the props drilling we can use the useContext

Parent <provider>

↓ child useContext

↓ child Use Contexts.

useReducer.

Const [state, dispatch] = useReducer( ) ;

---

useCallback will keep the function stored in their memory.

useMemo will keep the value stored in their memory.

Use Transition.

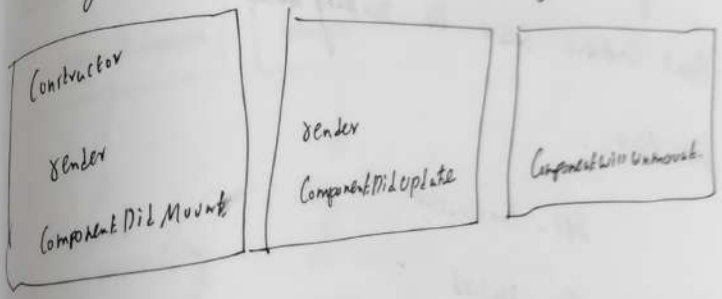
Syntax

Const [isPending, startTransition] = useTransition ();

↓

When we use this function that does not contains any of the high priority value.

React 5S Component lifecycle.



- Mounting → Component Created
- Updating → Updated via changes in props and states
- Unmounting → Component removed.

Component Rendering way.

1st Constructor → inside the class

↓

2nd Render → main Return of the class

↓

3rd ComponentDidMount → function inside the class.

ComponentWillMount() → deprecated

If a child component is being used inside the Parent component then the rendering method would look like...

App - Constructor ✓

App - Rendered ✓

Child Clock Component Render ✓

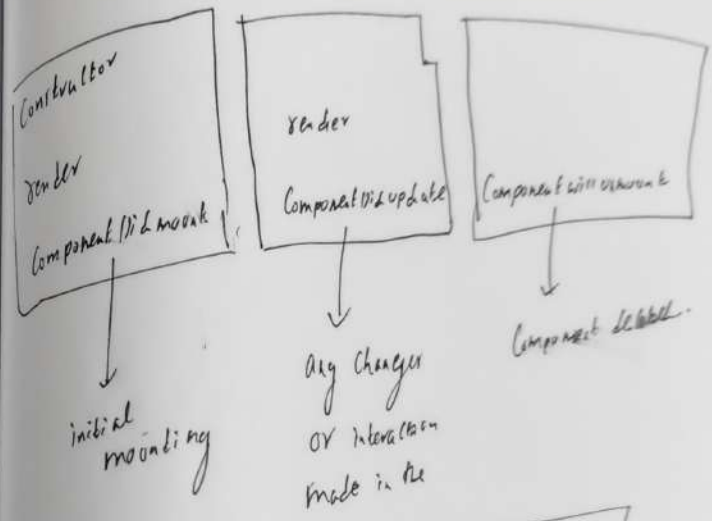
Child Clock DidMount ✓

App - Mounted ✓

Child Clock Component Render +++ ✓

ComponentDidUpdate.

↳ This function checks whether the user interacts with the application and shows how many times it is updated.



Lifecycle methods

ComponentDidMount()

ComponentDidUpdate()

ComponentWillUnmount()

Class based Component is where we can use the Constructor and Component lifecycle.

UseEffect()

To work on function based in react we can use useEffect().