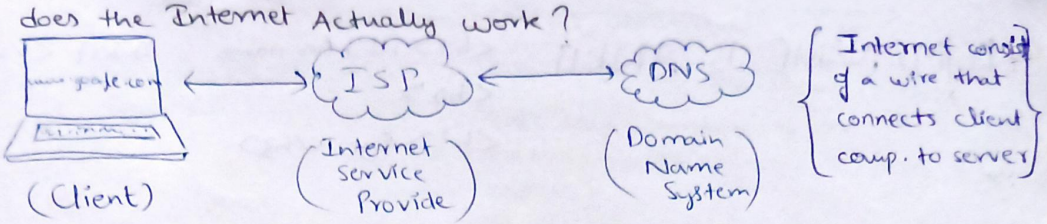
NAME: ANKIT VERMA STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: _____

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Web - Dev		
1)		Front-End Web Development		
2)		Introduction to HTML		
3)		Intermediate HTML		
4)		Introduction to CSS		
5)		Intermediate CSS		
		{2023 - Updated}		
1)		Introduction to HTML		
2)		Intermediate HTML		
3)		Multi-Page Websites		

FRONT-END WEB DEVELOPMENT

- How does the Internet Actually work?



When we type → 'google.com', then BTS (behind the scene), my browser will send that message to your ISP.
(these are the ppl. who you pay to be able to access the internet)

ISP then relay that message to DNS server, then DNS server looks in it's database to find the exact IP address of that website that you're trying to access.

- Every single comp. has IP address (kinda like postal code)

Once the DNS server finds the IP address, it will send it back to your browser through the ISP.

- How do websites work?

Now data that we receive from server usually consists of 3 types of files: HTML, CSS, JavaScript

- If a website was a House:

HTML would be the actual bricks of the house. Similarly, the HTML file contains the content of your website like the text content or the images or buttons or links

CSS would help in shape of door or colour of walls. The CSS file would determine how my website will look, like background of page & shape of button, helps in styling the font

JavaScript would be like adding light bulbs that can turn on & off. It allows my website to ~~actually~~ do things & become functional.

INTRODUCTION TO HTML (PRACTICE 1)

HTML = Hyper Text Markup Language

- Using HTML Tags

Go to → codepen.io → Start Coding → In HTML

<h1> Hi </h1> ---> HI

<h2> Hi </h2> ---> HI

<h6> Hi </h6> ---> HI (tells that - h6 code ends after HI)

Suppose if we want to add a line break → then write →
 (break)

{ < > } inside tag

(Self-closing tag)

Ex: In website | In HTML

HI
MY NAME IS ANKIT

Bye

<h1> HI ! My name is Ankit </h1>

<h3> Bye </h3>

To refer more → devdocs.io

• The Anatomy of an HTML Tag

Now if we want to add horizontal lines in our website, we use → <hr> (horizontal rule)

Ex:
<h1> HI </h1>
<hr>
<h3> Bye </h3>

HI
Bye

Now: there are diff. attributes for 'hr' → ex: align, color, noshade, size

Ex: <hr size="3" noshade>
T (space) (space) HTML attribute
HTML element

<> = (opening tag) width
 ...
will change whole text to centre
closing tag = </ >

⇒ Comments: <!-- text --!>
↳ Ignored by browser

• What is the HTML Boilerplate?

In VS code ~ ___.html ← (How to create a file for HTML)
↳ Short cut → !
↳ we will get : emmet abbreviation ✓

THE HTML BOILERPLATE

```
<!DOCTYPE html>                          (1)
<html lang="en">                         (2)
  <head>                                 (3)
    <meta charset="UTF-8">               (4)
    <title> Website name </title>        (5)
  </head>                                (6)
  <body>                                 (7)
    <h1> Hello World! </h1>              (8)
  </body>                                (9)
</html>                                  (10)
```

~ HAMBURGER
(3) Lettuce, Cheese
(5) Cream
Lettuce
Tomato
Meat
Tomato

(1): <!DOCTYPE html> : tells the browser that we have written our code in HTML 5 (latest version)

(2), (10): <html lang="en"> ... </html> : this is the root of the document
* this attribute says: language of the text content in the element (like for blind people, they have screenreaders)
* Inside <html> only, all the stuff will be written

(3), (6): <head> ... </head> : this is an area where imp. info about our website is placed that is not going to be displayed to the user.

(4): <meta charset="UTF-8">: * ensures that characters that you're using on your website gets displayed correctly (ex. ₹, ±, emojis can be shown as per our meta charset)
(charset = character setting; UTF-8 = default)
* This is convention, so let it be like that

(5): <title> My Website </title> : this title is displayed at top in the Tab bar

(7), (9): <body> ... </body> : * this is body element, where we will create & write our website
* this is where all of the content of website goes {texts, titles, images, links, ... etc.}

(8): content

In VS code: '!' →

<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge"> (just keeps our code compatible with Internet Explorer; we can delete this line as we don't use IE anymore)
<meta name="viewport" content="width=device-width, initial-scale=1.0"> (tells how website should be displayed relative to the screen that it's being rendered on.)

* We add attributes by giving it a name, and then after an equal sign, we give it a value

• How to Structure Text in HTML?
→ Right click on our file in VS Code → Copy path → Paste in google to (See our website)
(OR)
Click on bottom → Go live

* ... → paragraph tag

* Writing in italics → ... (only italics) & ... (italics + emphasis there) ↑ (better to use)

* → bold writing

ex: <body>
<h1> Hi </h1>
  My name is Ankit Verma

• HTML Lists

* To write in bullet points → (unordered list)
 ...
 ... (list items)

ex:
 Hi

→ • Hi

* To write in no.'s → (ordered list)
 ...

→ If we want 'ol' to start at particular no. ⇒ <ol start="7">
 ...

→ If using roman numerals ⇒ <ol type="i">
 ...

• HTML Image Elements

<img src="image address / image name">
(HTML element) (HTML attribute)
(if from browser) (if from folder)

In browser (right click on image to get its address)

(In VS code):
<body>
<img src="____" alt="____">
(alternative text) (will appear if image unable to load) → ①

If we want to resize image → <img src="..." height="..." width="...">

① If interested in Search Engine Optimization & getting your website ranking for certain key words, this is something that Google looks at to try & figure out what your webpage is about

* Now to remove our reliance on web, we can also put image into our folder (where HTML documents are saved)

* If we want to crop our image in circle → crop-circle.imageonline.co

ex: My folder → Coding > Web-dev > Front End Web Development → images
(where all my HTML codes saving) (3 subfolders) → pdfs
→ Intro. to HTML

In VS code, to render image, {Let image name = ankit.jpg}
<img src="images/ankit.jpg" alt="Myself">

• HTML Links & Anchor Tags

This is the part where we learn 'HT' (hypertext) of HTML.

 Hello → anchor tag
(HTML element) (HTML attribute) (hyperlink reference) (Link text)
(This is the website which our browser will take if we click on 'Hello')

* Shortcut → type 'a' in VS code, we will get!

* If we want to open more than 1 link, then press 'ctrl' button on keyboard & click on links.

* Now if we want a hyperlink which opens our part of website, then → ① Open a new file in your folder where you're saving all your html links.
② ex: → Let file name = contact.html
③ Then: Contact details

If this is inside a folder called Hi, then write as: "Hi/contact.html"

INTERMEDIATE HTML

• HTML Tables

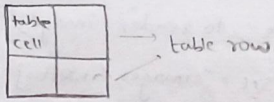
eg:
```
<h3> Work Experience </h3>
<p> 2005 - 2022 School Life <p>
<p> 2023 College Life </p>
```

Work Experience
2005 - 2022 School Life
2023 College Life

*Now to make it look more nice, we can create it as table!

eg:
```
<table>                          (type - t)
  <tr>                           (table row)
    <td> 2005 - 2022 </td>       (table data element)
    <td> School Life </td>
  </tr>
  <tr>
    <td> 2023 </td>
    <td> College Life </td>
  </tr>
</table>
```

2005 - 2022 School Life
2023 College Life



* In tables we also have headers, body & footers
  - <thead>
  - <tbody>
  - <tfoot>

eg:
```
<table>
  <thead>
    <tr>
      <th> Dates </th>          (by default its bold)
      <th> Work </th>
    </tr>                      (creates only 1 row)
  </thead>
  <tbody>
    I
  </tbody>
  <tfoot>                      (not used generally)
    <tr>
      <td> Statistics </td>
      <td> Now </td>
    </tr>
  </tfoot>
</table>
```

Dates	Work
2005-2022	School Life
2023	College Life
Statistics	Now

Now if we want borders of table also, then;

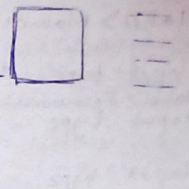
```
<table border = "1">           (This '1' is in pixels)
</table>
```

But table with this kind of border, it doesn't look good so much. Just managable!!

• Using HTML Tables for Layout

Now if we want to show our info. on right side of image, then we can use tables.

```
<body>
  <table cellspacing = "20">          (in pixels)
    <tr>
      <td><img src = " ___ " alt = " ___ "></td>
      <td><h1> Ankit Verma </h1></td>
    </tr>
  </table>
```



* Win + . (or) Win + ; → to get emoji's in Windows
  - . (full stop)
  - ; (semicolon)

• HTML Tables Code Challenge

Q) Write code → so output :

Skills

iOS Development ★★★★★	Photography ★★
Web Development ★★★★★	Painting ★

(We use nested table concept here)

A)
```
<h3> Skills </h3>
<table border = "1">
  <tr>
    <td>
      <table cellspacing = "10">
        <tr>
        <td> iOS Development </td>
        <td> ★★★★★ </td>
        </tr>
        <tr>
        <td> Web Development </td>
        <td> ★★★★★ </td>
        </table>
    </td>
    <td>
      <table cellspacing = "10">
        <tr>
          <td> Photography </td>
          <td> ★★ </td>
        </tr>
        <tr>
          <td> Painting </td>
          <td> ★ </td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

• HTML Forms

* We can create forms → (ex: Your name : [ ] / Your email : [ ] / [Submit])

```
<form>
<label> Name : </label>
<input type = "text" >
<label> Password : </label>
<input type = "password">
<input type = "Submit" >
</form>
```

(self closing tag)

Name : [Adcej] (we can type)

Password : [•••••] (be default it's masked)

[Submit]

{1} code : `<input type = "Submit" value = "10">` → [10]

* There are many kinds of input form in VS code

ex: date, range, email, colorpicker, --etc

• forms in Practice - Create a Contact Me Form

[Go to → contact.html]

(Worked in video, didn't work for me)

(we can also write another file name (in that same folder) so that after submitting it will go to that file (ex Angela Yu .html))

```
<body>
① <h1> My contact details </h1>
② <p> My Fictional Address </p>
③ <p> myemail@gmail.com </p>
④ <hr>
⑤ <form action = "mailto : (your gmail)" method = "post" enctype = "text/plain">
⑥ <label> Your Name : </label>
⑦ <input type = "text" name = "My Name :" id = " "> <br>
⑧ <label> Your Email : </label>
⑨ <input type = "email" name = "My email" id = " "> <br>
⑩ <label> Your Message : </label> <br>
⑪ <textarea name = "My message" rows = "10" cols = "30"></textarea>
⑫ <br>
⑬ <input type = "submit" name = " ">
⑭ </form>
</body>
```

{meanings}

⑤ `<form action = "mailto : syneirge2@gmail.com" method = "post" enctype = "text/plain">`

- action → (performs action)
- method = "post" → (means it posts user input to you)
- (so that o/p shows no symbol, only plain text)
- enctype = "text/plain" → before : Hi+Av / after : Hi Av

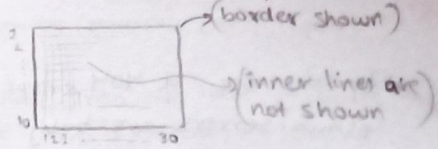
⑥, ⑧, ⑩ : In btw 'label', we should write — what we want from user

⑦, ⑨ : `<input type = "text" name = "My name" id = " "> <br>`

- "text" (or "email") → (in what format we want user to type)
- "My email" (or "My name") → (shown in o/p)

⑪ : `<textarea name = "My message" rows = "10" cols = "30"></textarea>`

- <textarea> → (creates a box like str.)
- name = "My message" → (shown in o/p)



⑬ `<input type = "submit" name = " ">`

- <input type = "submit"> → (shows submit button)

* After clicking on Submit, "action" starts.

∴ ex: (After Go Live)

My contact details

My Fictional Address

myemail@gmail.com

Your Name : [AV]

Your Email : [ankit@gmail.com]

Your Message : [Yo!] (we can resize)

[Submit]

{After clicking submit, my "Mail" opens up}

o/p : My Name = AV
My Email = ankit@gmail.com
My message = Yo!

• Publish Your Website!

* We use Github!

Steps: 1) Click on '+' (near your prof. pic) → new repository
2) Repository name = Project name (write it)
- Public
- Add a ReadMe File → Create repository

3) Now click on - Add file → Upload files (which are used in your code)
- Commit changes → write related to code / proj., as it will show our note (on outside)

4) Settings → Pages → Source → Choose: (main) → Save

5) Our website is now published & we can access it from anywhere
ex: https://ankit8125.github.io/Angela-Yu-CV/
https://(my username).github.io/(project name)/ ← (website link / Skeleton)

6) If we see a 404 error, then try after 30 mins as Github servers are busy

* Now, our main file (which is our homepage) is actually called - "index.html", and it should have a lowercase i & it should be spelled exactly like this!
This is very imp. because Github is going to look for a file with this name in order to serve as your homepage

7) Now if we see <DOCTYPE html> written on top of our website, then we can edit in - index.html & delete that

INTRODUCTION TO CSS
↳ (Cascading Style Sheets)

- Inline CSS

An example of website using CSS: https://web.archive.org/web/20180881920 2235js_/http://seanhalpin.io/

Now if we want to colour our website; then:

<body style="background-color: ___;">
(attribute) (CSS code)
- we can type colour name
- We can write hexadecimal code of that colour.
(type: colors css mdn) & {get} or (go to: colorhunt.co)

* Types of background color ways:
⇒ writing in hexadec. ways / color name.
⇒ rgba color model: denoted: rgba(, , ,);
r → red
g → green
b → blue
→ 255 corresponds to 100%.
→ Our lap. will automatically mix these colour variations & give us new color.
(Optional) a → alpha → no. btw 0 & 1 → (varies transperency) 0 = transparent, 1 = opaque

ex: <body style="background-color: rgba(120, 75, 75, 0.5);">

⇒ hsla model: h → hue → taking an angle of color circle (if no. written - then its in degrees)
s → saturation → 100 → completely saturated, 0% → grey
l → light → (100% - white, 0% - black)
a → alpha

⇒ background-color: current color; } changes whole website to black
: transparent; } changes whole website to transparent

- Internal Css

Now if we have to change colour of line, then we have to write background color -- at every line where <hr> is present, -- lengthy process.

So instead of that, we write → <head>

* In btw style tag, we have to specify name of element that we have to change & then we create a set of curly braces inside which we are going to write css code.

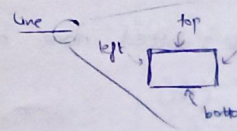
<style>
} the part which we wanna change in whole website at a time
</style>
</head>

ex: <style>
body {
background-color: blue;
}
hr {
background-color: white;
}
</style>

* Now basically a line isn't a line after all,
Line: ═══, rectangle with zero height
↳ (changes inside space as white)
* Now browser has some default CSS coding

for ex: (type → css default for browser) → in that → for <hr>: display: block;
margin-top: 0.5em;
margin-bottom: 0.5em;
margin-left: auto
margin-right: auto
border-style: inset
border width: 1px (pixel)

Now there are diff. border styles.
border-top-style: none; (no border)
hidden,
dotted:
Solid;
double;
groove;
ridge;
inset;
outset.
dashed;



* If border-style: none;
(then no borders will be!

* Everything that exist on webpage are essentially just boxes. & we can make those boxes appear → by using "pesticide" extension of chrome.

* We can also resize image →
```
<style>
    img {
        height: 200px;
    }
</style>
```
→ image will get resized.

* In <hr>;
```
<style>
    hr {
        background-color: blue;
        border-style: none;
        height: 2px;
        width: 100px;
    }
```
(we can also write as → width: 30%; which means it will make 30% of line based on our webpage size, so if we minimize our webpage, our line will also get adjusted according to it)

Q) Convert the line: ——— to ○○○○○ ?

A) If →
```
hr {
    background-color: white;
    border-style: dotted;
    height: 0px;
    width: 30%;
}
```
→ ○○○○○○○○○○

Now → border-style property:

(i) 1 value specified, it applies same style to all four sides

(ii) 2 values specified, 1st one applies to top & bottom
2nd one applies to left & right

(iii) 3 values specified, 1st → top, 2nd → left + right, 3rd → bottom

(iv) 4 values specified → top, right, bottom, left ← in that order it's applied (clockwise)

∴
```
hr {
    border-style: dotted none none;
}
```
(OR)
```
hr {
    border style: none;
    border-top-style: dotted;
}
```
→ ○○ ○○○○

So answer →
```
hr {
    border-style: none;
    border-top-style: dotted;
    border-color: grey;
    border-width: 5px;
    width: 5%;
}
```
(5 times)

- External CSS

Now suppose we want our 'hobbies' & 'contact me' page also to look like main page, so it would little length if we copy-pasted 'style' code in each file. So instead of that we create a new folder → Css → inside, we create new file → styles.css

Now we cut-paste from index to styles.css

(cut inside of <style> </style> → can delete)

And now whereever we want to apply; be it - index, hobbies, contact just write this piece of code in: <head>
```
<link rel="stylesheet" href="css/styles.css">
</head>
```

* To make the text colourfull;

→ then inside styles.css :→
```
h1 {
    color: blue
}
```
→ (changes all <h1> to blue colour)

- How to Debug CSS Code

* To check our code in chrome → right click → inspect
→ ⋮ → more tools → developer tools

Now, one case of debugging is:
```
<head>
    link rel="stylesheet" href="css/styles.css">
    <style>
        body { background-color: red; }
    </style>
</head>
<body style="background-color: white">
    ⋮
</body>
```
(let color pointing here = blue)

Now our website will show us white background. So to know what's happening
→ Click on inspect
→ In elements: select body →
→ Now at bottom, see styles
It's showing all of the CSS that is being applied to body of our webpage.

→ Now the chrome has default priority to arrange in this way.

(First 'body' ; then in 'style' ; then 'styles.css')

(inline CSS) > (internal CSS) > (external CSS)

→ We can untick the color that we don't want to show up on our webpage

This unticking & ticking on chrome, affects how the webpage gets displayed. {our code will not change}

So if we hit 'refresh', it goes back to how it will be displayed across all the browsers. So change is done locally.

* This means, we can apply a global CSS rule to all of our webpages, but on the individual web pages, we can apply more specific rules using internal (or) inline CSS as more or less one-off changes for that specific page or that specific element on that page

- The Anatomy of CSS Syntax

selector { property : value } ; ex: h1 {color : red} ;

("who" do you want to change) ("what" you want to change) ("How" how much we want to change)

* Now suppose in styles.css → h1 {color red};
then VS code will show 'problems' notification at bottom, so that we can resolve it.

* h1 {
color : red;
font-size : 200px;
}

Now the best practice is to write all the properties in alphabetical order, so that it would be easy for us to solve the debugging

* You can refer to CSS mdn references to know all about the keywords (effectively our property)

- CSS Selectors

To comment out in CSS : /* */

* Go to → emojipedia.org → for emojis

Now suppose we want to give 2 diff. backgrounds to 2 diff. images, then:

In index.html : <img class="Hi" src="..." alt="image">
<img class="Bye" src="..." alt="images">

Now in styles.css :

/* (TAG SELECTORS) */

```
img {
    background-color : red;
}
```

/* (CLASS SELECTORS) */ (add a full stop & write class name)

```
.Hi {
    background-color : blue;
}

.Bye {
    background-color : green;
}
```

→ So now : class selector would be given first preference to img. background & if nothing is mentioned in img src, then the tag selector would be given preference

- Classes vs Ids

(intern) → ex: <h1 id="heading"> Yo </h1>

(now in styles) (refers to text colour)

/* (ID SELECTORS) */

```
#heading {
    color : blue;
}
```

This changes "Yo" to blue colour & as usual ID selectors are given more preference than tag selectors

∴ Preference order = ID > Class > Tag > Chrome default style

* Now id="heading" can be used only once in that html page whereas with class, it doesn't really matter. So if we were to create another id, then it should have some diff. name other than "heading", whereas class can have same name.

* Use classes when you want to apply same style to group of related items & use the id to apply a specific style to a single element on your web page.

* Any html element can have more than one class but it can only have 1 id.

Ex: (index): <body>
:
<img class="Bye circular" src="..." alt="..." >
</body>

(styles.css):

```
.Bye {
    background-color: green;
}

.circular {
    border-radius: 100%;
}
```

Then our image will become circular & have its background color as green.

But if we try to do the same with id's:

(index): <body>
<h1 id="heading big">Yo</h1>

(styles.css):

```
#heading {
    color: blue;
}

#big {
    font-size: 1000px;
}
```

Now our webpage will neither be blue in color nor font size will be 1000px.

Since the format is incorrect it doesn't read this code & then directly reads tag selector to check if any property is mentioned or not & makes changes accordingly

→ In CSS reference mdn → we can see few keywords having a little colon in front of them & these are called: pseudo classes
(Ex: let us look at :hover)

(styles.css)

```
img:hover {
    background-color: gold;
}

.Hi {
    background-color: blue;
}

.Bye {
    background-color: green;
}
```

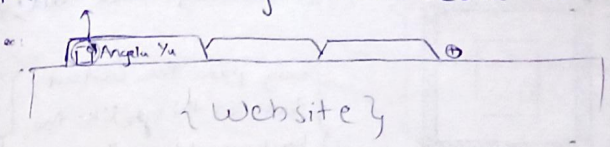
So now if we move our cursor to image in class 'Hi', it's background color will change from blue to gold & once I remove my cursor from the image, it will change back to blue. Similarly with image in class -Bye

∴ The ':hover' helps in changing the color when cursor points at it.

INTERMEDIATE CSS (PRACTICE 2)

• What are Favicons?

Favicons are images shown beside our website in tag region.



→ We can go to "favicon.cc" to make our own favicon. After making, download favicon in the same folder where our code is present.

↳ (In VS code): <head>
:
<link rel="icon" href=favicon.ico>
</head>

Now we can see our favicon on our website.

• HTML Divs

*Pesticide" extension for Chrome highlights all of my HTML elements & essentially all the boxes that are on screen.

(Ctrl + mouse pointer) → mouse points on a part of website → Pesticide shows what type is it (in blue colour)

* Div stands for content division element, & it basically allows you to split up & divide your content into seperate containers/boxes so that you can affect the layout of box seperately

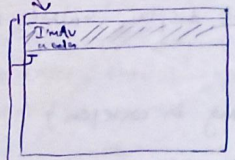
(index) →

```
<body>
    <div class="">
        <h1>I'm AV</h1>
        <p>a coder</p>
    </div>
</body>
```

(Styles) →

```
div {
    background-color: blue;
}
```

(webpages)



(blue colours)

Now to reduce this spacing

(styles):

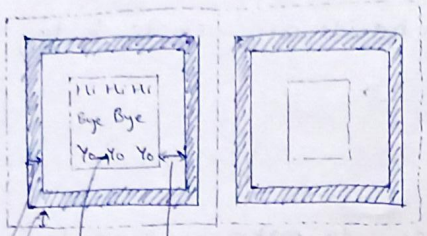
```
body {
    margin: 0;
}

h1 {
    margin-top: 0px;
}
```

(when 0, no need to specify unit)

The Box Model of Website Styling

* Border :→ {border-width: 0px 10px 20px 30px;}
  (pen clockwise direction)
  & ex: border: 30px black solid;



(border) (Content) (padding) → creates some space for content all-around.
↳ doesn't affect content size
(margin)
↳ helps differentiate btw 2 boxes (if adjacent)

{only pads the content, but not things like the background image/background color.}

Ex: Look/Make: 


```
(index)
<body>
    <div class="top-container">
    </div>
    <div class="bottom-container">
    </div>
</body>
```

```
(Styles)
.top-container {
    background-color: red;
    width: 200px;
    height: 200px; border: solid 20px;
}
.bottom-container {
    background-color: blue;
    width: 200px;
    height: 200 px; border: solid 20px;
    margin-left: 240px;
}
```

o/p →

CSS Display Property

It has 4 display values: → Block, Inline, Inline-Block, None

(Now checking in codepen)

* Block elements are those that take up essentially the whole width of the screen on a web page (we can use pesticide & check), so effectively blocking out any other elements from sitting next to it on the left or on the right.

→ Common block elements:
- Paragraphs ( )
- Headers (<h1> through <h6>)
- Divisions ( )
- List & list items (<ol>, <ul>, &)
- Forms (<form>)

ex: a programmer {Suppose we want to underline "pro"}

↳ a pro grammer.

(styles) → .pro { text-decoration: underline; }

(using pesticide) → a programmer
↓
(pro is still in the same line)

* An Inline display element only takes up as much space as it needs to in the height & in it's width so you can see that box around span is only as large as it needs to be!

→ Common inline elements:
- Spans ()
- Images (<img>)
- Anchors (<a>)

{ex: In practice 1 → that is why: My Hobbies Content We were side by side}

- Hello & p { background-color: red; } → Hello (box in red)

 Hello & span { background-color: blue; } → Hello (Only this box in blue)

(automatically goes to next line)

(If span & para in same line ↳ span) para (preference)

→ In inline elements, we can't change the width of all the spans, but if we specify width to block elements, then it complies (But it still doesn't let any other element sit on the same line)

* We can change the display property of any given element:

ex: for paragraph element

↳ p { background-color: red; display: inline; } → Hello
→ (now can't change width)

* Inline-Block elements: → we can change width
→ we can make them go onto same line.

And this is kind of what image elements are like: inline elements (displayed next to each other) & inline-block elements (can change it's size)

* None - simply removes that element from website, as if it didn't exist

&

Visibility - makes that element disappear but it keeps it's original position, & all the other elements still flow around it as if it's still there.

ex.
```
<p> Hello </p>
<p class="second-p"> World </p>
<span> Hello </span>
<span> World </span>
```

(style)
```
p{
    background-color: red;
    width: 100px;
    display: inline-block;
}
span {
    background-color: blue;
    width: 100px;
}
```

```
.second-p {
    display: none;
}
```
→ Hello | Hello World

(OR)

```
.second-p {
    visibility: hidden;
}
```
→ Hello | Hello World

• Learn More About Web Design

Typeface: Name of the Design in family of styles
ex: Arial, Serif

* San-serif family looks so much friendly, more approachable, more novel and more contemporary

((Try to make stuff using 2 fonts in general) ....

ex: Heading → Serif
Body → San-serif
or vice versa } → just creates more interest in my design.

• CSS Static and Relative Positioning

Without CSS, our HTML element already has predefined rules for how it should be displayed on my webpage even if we don't have CSS
Let's take a look at these rules

1) "Content is Everything": Block elements even though they take up 100% of width, height is still determined by the content
So: my content is the first thing that determines how large things get displayed & what the height & the width will be. And this is despite any CSS.

2) "Order Comes From Code"

3) "Children Sit On Parents": By default our HTML elements that are children, they will sit on top of our parent element

ex:
```
<div>
  <h1>a <span>pro</span> grammer</h1>
</div>
```

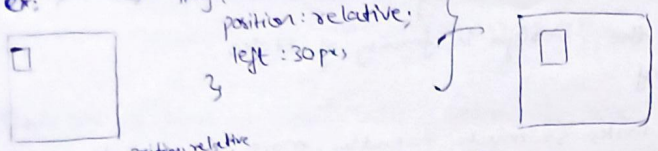
Now we can also set CSS property ~ position property in order to position elements on screen the way that we want instead of just going along with the default layout.

(Relative, Static, Fixed, Absolute)

1) Static: All HTML elements are static in their position by default & static just means go along with HTML rules.

2) Relative: Allows us to position the element that we select relative to how it would have been positioned had it been static/default.

ex: img { position: relative; left: 30px; }



position relative right; position relative top; position relative bottom; position relative left;

when we move an element that has a relative position it doesn't affect the position of anything else on screen.

Now ex: try to convert  to 

(HTML) ①

```
<body>
  <div class="red">
  </div>
  <div class="blue">
  </div>
  <div class="yellow">
  </div>
</body>
```

(CSS) ②

```
.red {
  height: 100px;
  width: 100px;
  background-color: red;
  position: relative
  left: 200px
  display: inline-block;
}
```

③

```
.blue {
  height: 100px;
  width: 100px;
  background-color: blue;
  display: inline-block
  position: relative
  right: 100px
}
```

④

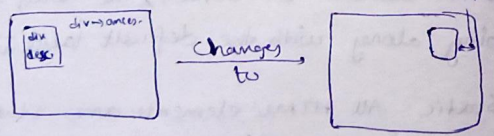
```
.yellow {
  height: 100px
  width: 100px
  background-color: "yellow"
  display: inline-block
  position: relative
  right: 100px
}
```

If position (right/left) not written, then order: R B Y

Now reason why spaces btw our squares are inconsistent is because by making it an inline block there's actually a little space that gets added in by the browser.

- Absolute positioning

ex: div { position: relative; } img { position: absolute; right: 30px; }



position relative to nearest positioned ancestor or top left corner of webpage.

In absolute positioning, we are positioning the element, so in this case the image element, relative to it's parent, & in this case its parent is that red div

{In most cases, the parent is simply the entire body of the webpage}

* Relative positioning means, we are adding a margin relative to where the element should have been.
But absolute position means that we are adding a margin to its parent element.

ex: ① ② ; position: absolute; left: 200px; — ⑤ ③ ④

when ⑤ not written: R Y

when ⑤ written: B R Y

By using AP (absolute positioning) it does affect the flow of our HTML. When we are using this, we are actually taking the element out of the flow of document & it's no longer considered a part of the natural flow of the document.

So that's why the blue square & yellow square have shifted up, because it's as if I've just gone & deleted that div. Basically red is now dead to blue & yellow

[body { margin: 0px; }] → means now all elements will stick to the exact sides of webpage

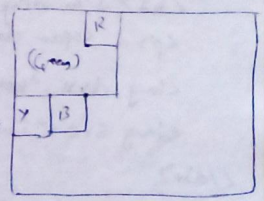
Now the parent doesn't have to be the body (always). It can also be the closest parent that has a relative layer

ex: ①

```
<body>
  <div class="container">
    <div class="red">
  </div>
  :
</body>
```

(CSS)

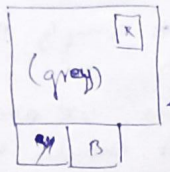
```
.container {
  position: relative;
  width: 300px;
  height: 300px;
  background-color: grey;
}
```



⑤ → left: 100px; ④ {right ← deleted} ⑤ → {position: absolute;} → ⑥

Now our red square is a child of that parent container. And now it's defining its position relative to that grey container square.

If ⑤ → { left: x; right: 20px; }



Blue & yellow are still relative to the body because we haven't defined a vertical position for it, it's being pushed down by this element with a relative layer.

Can change → write → top: 0;

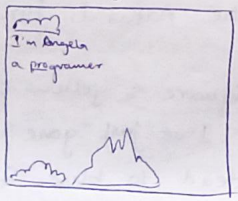
* Now if ".yellow"

.yellow {
position: fixed; → (position relative to top left corner of browser window)
top: 0;
}

then if I scroll through the webpage it will stay in its current position (useful when we have → nav bar / side bar)

- The Dark Art of Centering Elements with CSS

Now → my website :



To center elements → we use : text-align property inside our parent container

(HTML)

```
<body>
<div class="top-container">
    <img class="top-cloud" src="___" alt="___">
    <h1> I'm Angela </h1>
    <p> a <span class="pro">pro</span>grammer.</p>
    <img class="bottom-cloud" src="___" alt="___">
    <img src="images/mountains.png" alt="___">
</div>

</body>
```

(styles.css) * In order to position cloud relative to something, one of its parents has to have it's position set as relative. And if we don't then it will be relative to the body.

So, we want position of cloud relative to top-container.

```
body {
    margin: 0px;
    text-align: center;
}

h1 { margin: 0 auto;
}
```

```
.top container {
    background-color: ...;
    position: relative;
    padding-top: 100px;
}

span { text-decoration: underline }
```

```
.bottom-cloud {
    position: absolute;
    left: 300px;
    bottom: 300px;
}
```

```
.top-cloud {
    position: absolute;
    right: 300px;
    top: 50px;
}
```

- Font Styling in Our Personal-Site

Sans-serif : F Serif : F

For most browsers, the default serif is Times font, & the default sans serif is Arial

ex:
```
body {
    font-family: verdana, sans-serif;
}
```

Now if brower / OS that the user's using doesn't have this "verdana" font installed, then it will default to whatever is the sans serif font that is installed on their system.

* To ensure that everybody has the same viewing experience regarding fonts on my website, we can use - font embedding

↳ go to: fonts.google.com

② Now select any font style
② + Select this style

③ Click on top right: ▪▪ ▪+

④ Copy the whole link

⑤ Paste inside: <head>

<link ... >
</head>

⑥ In that link only (at google), bottom we will find CSS rules to specify. Copy that

⑦ Now paste in styles.css under whatever heading you wish to change the font.

This specifies a location where if users don't have these fonts already installed on their system, then the browser will take them to this location & grab these fonts for them

• Adding Content to Our website

* Flaticon.com → to get images
* Giphy.com → to get gifs

• CSS Sizing

(styles.css)

```
h1{
  font-size: 16px;
}
```

16px = 100% = 1em

(static) → means even if we change size of our webpage, our font size won't change

(dynamic) → adjusts itself as per webpage size

If →
```
body{
  font-size: 2em;
}
h1{
  font-size: 1em;
}
```
→ I'm Angela } 3em

the parent's font-size (body) gets added to it's elemental size (h1), when size is dynamic

If →
```
body{
  font-size: 20px;
}
h1{
  font-size: 90px;
}
```
→ I'm Angela } 90px (static size doesn't depend on any other sizes)

Now in order to avoid addition of parent size on its element, there is something called - rem (root em), so that means ignore all of the parent settings for the font size & just set it to this relative to the root.

```
body{
  font-size: 200%;
}
h1{
  font-size: 5.625rem;
}
```
→ I'm Angela } 5.625rem

: CSS Font Property Challenge Solutions

To change colour of font → (styles) →
```
h1{
  color: blue;
}
```

* font-weight : specifies the weight / boldness of font.

```
ex: h2{
  font-weight: normal;
      ↳ bold
      ↳ lighter;
}
```

* line-height : ← used to set the distance btw lines of text

```
ex: h1{
  line-height: 1;   → (no units*)
}
```
→ (1 → default)

• CSS Float and Clear

index:
```
<div class="skills">
  <h2>My Skills.</h2>
  <div class="skill-row">
    <img class="image" src="..." alt="..." >
    <h3>Design & Development</h3>
    <p class="hi">  ---  ---  </p>
  </div>
  <div class="skill-row">
    <img class="image2" src="..." alt="...">
    <h3>Hot wings Challenge</h3>
    <p class="hi2">  ---  ---  </p>
  </div>
</div>
```

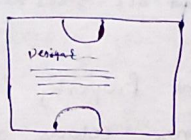
Now as we can see, the text takes 100% of space/width of webpage, so let's reduce it to 50%.

(Styles) →
```
.skill-row {
    width: 50%
}
```

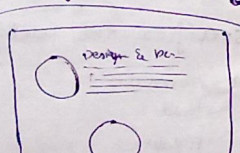

Now in order to re-arrange it to centre.

```
.skill-row {
    width: 50%
    margin: auto;
}
```
(auto helps to create symmetry)

Now to make our text left-aligned

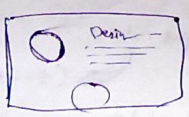
```
.skill-row {
    text-align: left;
}
```


Now to wrap our text beside image, we use "float" property

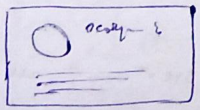
```
.skill-row {
    float: left;
}
```


(left of text will be floated, so that image fits in)

Let's give little more spacing btw image & text, to make it look more beautiful.

```
.image {
    margin-right: 30px;
}
```


Now, for example, if we wanted that title to wrap around this image & to be on the right of the image, then we can use another property to make sure that this text does not wrap & end up at the bottom over here below the image, & that property → "clear".

```
.hi {
    clear: left;
}
```


{clear kinda opposite of float}

* 'padding-inline' → means adding space on both sides

• Stylised Personal Site Solution Walkthrough

Now links present at bottom → ex. LinkedIn Twitter

By default, all anchor tags have an underline that shows you that this is clickable

Now, we can get rid of that underline by changing text-decoration

(Styles) →
```
a {
    text-decoration: none;
}

a:hover {
    color: white;
}
```
{colour changes when mouse points on this}

* 2023-Updated *

Introduction to HTML

• Self Closing Tags

⇒
```
<p> Hi </p>
<hr />
<p> Bye </p>
```
Hi

Bye

{<hr /> horizontal rule element}

[<hr /> or <hr></hr> → both correct]

Now → 'Break' element

```
<p>
To see World <br />
Heaven in Flower <br />
Infinity in hand <br />
Eternity in hour <br />
</p>
```
To see World
Heaven in Flower
Infinity in hand
Eternity in hour

→ Generally 'Break' element used for poems & stuff but if we want to go to next line → use new paragraph tag.

Intermediate HTML

• Anchor Elements

HTML attributes

Skeleton: <tag attribute=value another attribute=value> Content </tag>

eg: This is Google → means after selecting that text, we can also drag it.

- Image Elements

ex: `<img src="https://picsum.photos/200" alt="forest" />`

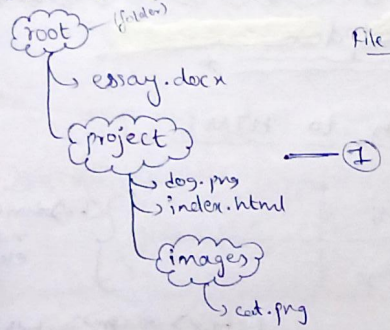
{ After a forward slash, we can add in the size of our image. So in this case we want a square of 200px x 200px. }

Multi-Page Websites

- Computer File Paths

A file path can direct someone to a file / folder.

- Absolute file path:



File path → C:/Project/Images/cat.png

- Relative file path:

If we consider "index.html" as our starting point, then relative to it how do we get "cat.png"?

↳ 'images/cat.png'

→ `./dog.png` ↳ it tells to stay within the current directory & look over there (the folder that your file is directly located in.)

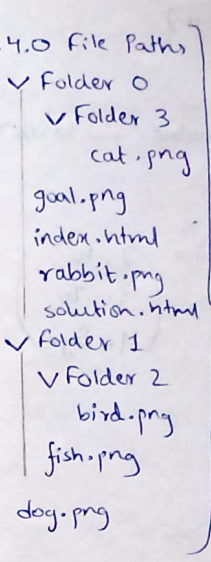
(can also omit ./)

`../essay.docx` ↳ it just means go up a level & then go into whichever resource/path you want to navigate

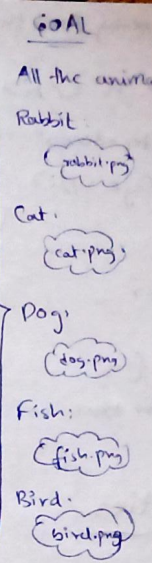
Here starting at index.html in project folder as this is where our starting point is. If we go level up using '../', then it enables us to go in root folder. Now relative to root folder, in order to navigate to essay.docx, we have to write → ../essay.docx

Now:

ex:



GOAL



SOLUTION

```
<h1> All the Animals </h1>
<h2> Rabbit: </h2>
<img src="./rabbit.png" alt="Rabbit">
<h2> Cat: </h2>
<img src="./Folder3/cat.png" alt="Cat">
<h2> Dog: </h2>
<img src="../dog.png" alt="dog">
<h2> Fish: </h2>
<img src="../Folder1/fish.png" alt="fish">
<h2> Bird: </h2>
<img src="../Folder1/Folder2/bird.png
alt="bird">
```

We will mostly be using relative paths.

At end of our website → we can add a footer element with our name & any copyright info. (or) other disclaimers.

ex:

```
<footer>
    <small> © Angela Yu. All rights reserved </small>
</footer>
```

Introduction to CSS

- CSS Selectors

↳ Attribute selector ex.

```
<p draggable="true"> Drag me </p>
<p draggable="false"> Don't drag me </p>
```

Drag me

Don't drag me

(styles.css)

```
p[draggable="false"] {
    color: red;
}
```

(red color)

&

Universal Selector ex: `* { color: red; }` → styles.css

(means → select all)

(start numbering from 3 there)

```
<ol>
  <li class="note" value="2"> Hi </li>
  <li class="note" id="id-selector-demo" value="3">Yo</li>
  <li class="note" value="4">Bye </li>
</ol>
```

o/p →
2. Hi
3. Yo
4. Bye

→ to change 'Yo' to blue colour

(styles.css)

```
li[value="4"] {
    color: blue
}
```

CSS Properties

- Font Properties:
  - font size: 1px → $(\frac{1}{96})^{th}$ of inch ; 1pt → $(\frac{1}{72})^{th}$ of inch
  - 1em → 100% of parent
  - 1rem → 100% of root (relative to <html> </html>)

ex:

```
<html> ——— (1)
  <body>
    <h1> Hello </h1>
    <footer> ——— (2)
      <h2> Hello </h2> ——— (3)
    </footer>
  </body>
</html>
```

* If (2) → size = 20px
  (3) → size = 2em = 2x20 = 40px

* If (1) → size = 30px
  (2) → size = 20px changes to 50px
  (3) → size = 2rem (independent of (2)) → = 2x30 = 60px

Font Weight: normal, bold → Keywords; lighter, bolder → Relative to parent; number → 100-900

Font family → if font has many words in it, then write in inverted commas

```
h2 {
    font family: "Times New Roman", Serif ;
}
```

- Right Click → Inspect → we will get the coding for that website (on any website)

Text transform property:

(style.css)

```
h1 {
    text-transform: uppercase ;
}
```

(to lowercase or full-width)

Intermediate CSS

- The Cascade - Specificity & Inheritance

There are 4 broad categories which we look at when we're determining the overall level of importance of CSS Rule

→ Position → Specificity → Type → Importance

* POSITION: ex:

```
li {
    color: red;
    color: blue;
}
li { color: green ; }
```

the more lower, more preference is given (∴ Color would be shown as green.)

* SPECIFICITY: ex: <li id="first-id" class="first-class" draggable>

```
li { color: blue; }                  ——— element (4)
.first-class { color: red; }         ——— class (3)
li[draggable] { color: purple; }     ——— attribute (2)
#first-id { color: orange; }         ——— id (1)
```

∴ Most priority given to 'id' ↳ ∴ color shown is orange

* TYPE: ex:

```
<link rel="stylesheet" href="./styles.css" > → external (3)
<style></style> ——— internal (2)
<h1 style=" " >Hello</h1> ——— inline (1)
```

first we check any styles relevant from external stylesheet, then we check whether if there are any from internal stylesheet & finally if there are any inline. (most priority)

* IMPORTANCE: ex:

```
color: red;
color: green !important;
```

(Keyword)

{ ensures that this is going to be most imp. rule relative to that element. }

!important → doesn't matter how imp. this prev. rule is, it could have been inline (or) it could have been in an id-selector, or it could have been at very bottom of page. This rule is going to come out as the Top Trump

ex: `<h1 class="a-class another-class">Hello</h1>`

```
.a-class {
    color: green;
}
.another-class {
    color: blue;
}
```

(as it comes next & last, so this is more prior)

• Combining CSS Selectors

(index) → `<div class="box inner-box"> <p>white</p> </div>` (styles) →

```
.inner-box p {
    color: white;
}
```

as long as it's closed inside the div, it counts as a descendant & we're looking for a paragraph

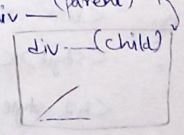
Group = apply to both selectors (ancestor) seperated by comma.

ex: `selector, selector { color: violet; }` (descendant)

(this relationship only one generation deep)

(parent) (child) (direct child)

`selector > selector { color: firebrick; }`



in order to select that particular element

* It has to be only one level nested.

ex:
```
<div class="box">
    <p class="done">Hi</p>
    <ul class="list">
        <li>Wash</li>
        <li class="done">Read</li>
    </ul>
</div>
```

To change 'li' elements to blue color.

```
.box li {
    color: blue;
}
```

Chaining = apply where all selectors are true → used to ↑ specificity

```
selectorselector {
    color: green;
}
```

ex: `<h1 id="title" class="big">Hello</h1>`

```
h1#title.big {
    color: green;
}
```

(always start with element if changing)

* There is no space btw selectors.

Combining combiners

(ancestor) selector (chain) selectorselector {

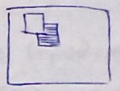
```
    font-size: 2rem;
}
```

• CSS Positioning

Z-index → determines which one goes on top of which

↳ default = 0

ex: (in styles) ↳ `z-index: -1;`

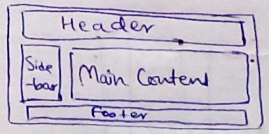


↳ `border-radius: 50%;` → (changes box to circle)

Advanced CSS

• CSS Float

We use 'float' property on images → (in styles.css)



• How to Create Responsive Websites

Main ways to create responsive websites using CSS

1) Media Queries
2) CSS Grid
3) CSS Flexbox
4) External Frameworks ex. Bootstrap.

• Media Queries

ex: Skeleton: `@media (max-width: 600px) { h1 { font-size: 15px; } }`

(target: small screen)

(it will override the general code)

→ define what called break point. It says that at this particular width, anything that is less than or equal to 600px, then should use this styling

ex: `@media (min-width: 600px) { h1 { font-size: 15px; } }`

(target: large screen)

{ Anything that is from 600px upwards, then we should have a diff. styling }

600px

ex: `@media (min-width: 600px) and (max-width: 900px) { /* Styles for screens btw 600px & 900px */ }`

600px 900px

ex: `@media (max-width: 600px) and (min-width: 900px) { /* Style for screens <600px & >900px */ }`

600px 900px

→ we write this in styles.css

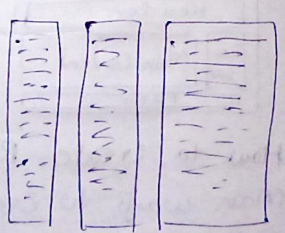
Flexbox

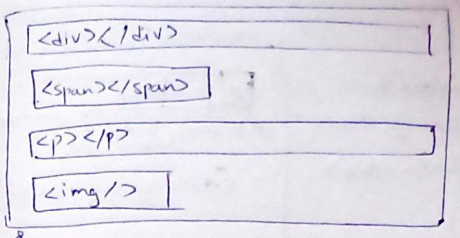
• Display: Flex

```
<div class="container">
    <div class="one"> <p> — <p></div>
    <div> <p> — <p></div>
    <div> <p> — <p></div>
</div>
```

(styles.css)

```
.container {
    display: flex;
    gap: 10px;
}
```



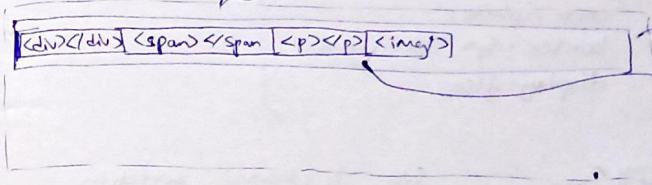


(website)

→ when we use flexbox, we enclose all the elements that we want to display inside a flexible container, then what happens is all of these previous display values will be ignored.

And instead were going to have everything laid out by flexbox & the width of each of the elements inside the Flexbox will normally be based on the content size

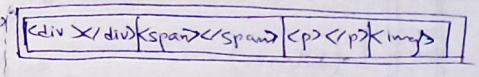
{doesn't matter what type of element is inside flexbox}



(like this)

→ we can use 'gap' property to give space btw them

if → display: inline-flex

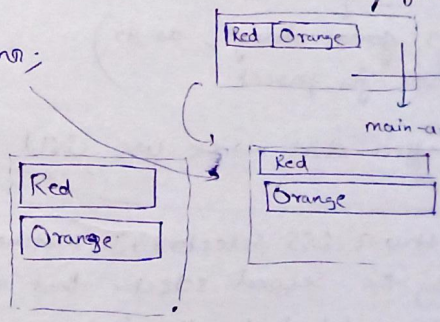


• Flex Direction

flex-direction: column;

flex-basis: 100px;

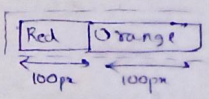
(it's the height)



flex-direction: row;

if → flex-direction: row;
flex-basis: 100px;

(it's the row)

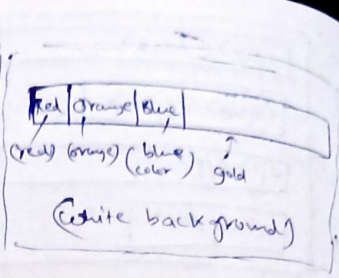


* flex-basis is set on child element

ex: <body>

Red
Orange
Blue

</body>

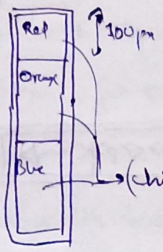


(styles.css) → .red { background-color: red };
.orange { " : orange };
.blue { " : blue };

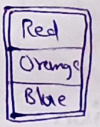
.container {
color : white;
border : 5px solid gold;
display : flex;
}

① ·

Goal. →



Soln. To change direction
↳ .container {
flex-direction: column;
}
display : inline-flex;
}



* If we write → flex-basis: 100px; in ①
↓
(nothing's gonna change, as it's not used for parent)

Now to affect child → we use child combinator (>)

'>' placed btw 2 CSS selectors, It matches only those elements matched by the second selector that are the direct children of elements matched by the first.

*
↳ CSS universal selector
↳ we can select all the elements
↳ only children
(not grandchild, — etc)

ex: Red
(not selected by *)

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks

Web - Dev

1) Front-End Web Development
2) Introduction to HTML
3) Intermediate HTML
4) Introduction to CSS
5) Intermediate CSS

{2023 - Updated}

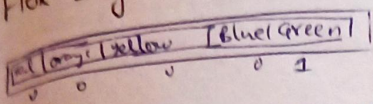
1) Introduction to HTML
2) Intermediate HTML
3) Multi-Page Websites
4) Intro to CSS → CSS Properties → Intermediate CSS → Advanced CSS
5) Flexbox
6) Grid
7) Bootstrap
8) Introduction to Javascript
9) Jquery

Node JS

Express JS with node js

(style) .container > * {
flex-basis: 100px;
}

Flex Layout



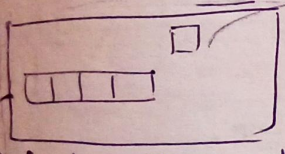
(we can set other colors order also)

Now we do ⇒ .blue {
order : 1
}

'blue' box goes to end of container.

- flex-wrap : nowrap → ele's will get added on main axis only
  flex-wrap : wrap → ele's will go to next line if enough space is not present on above line

- justify content → flex start
  → flex end
  → centre
  → space between
  → space around
  → space-evenly



If we want this to be seperate
↳ align-self : flex-start

align-items: center
height : 70 vh
(viewport height)

Flex Sizing

Priority : Content width < Width < flex-basis < min-width/max-width

- flex-grow : 1; → when tab size ↑'s, flex size also grows
  flex-shrink : 1; → when tab size ↓'s, flex size reduces

if set to 0, it won't grow (or) shrink & if we try to reduce our tab, content will be lost.

⇒ flex : 1 1 0 ; (or) flex : 1 ;

grow shrink basis

∴ grow = shrink = 1
basis = 0

* list-style : none ; (removes bullet points)

GRID

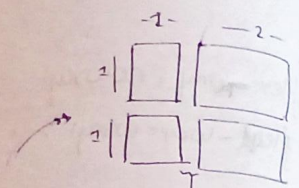
- Display : Grid :

Flexbox → generally for 1D arrangement
Grid → generally for 2D arrangement

ex: class: "container"
<p> ... <p>
<p> - - -
<p> -
<p> -

→ in CSS

```
.container{
  display: grid;
  grid-template-columns : 1fr 2fr;
  grid-template-rows : 1fr 1fr;
  gap : 10px;
}
```



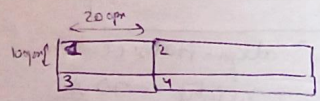
fr = fractional ratio → 1fr 2fr = 1 : 2 (see column & rows)

- Grid Sizing

shortcut:
grid-template : 100px 200px / 400px 800px ;
denotes rows denotes columns

1) We can use 'rem' sizing.

2) grid-(r--)-(r--) : 100px auto ;
grid-(c--)-(c--) : 200px auto ;



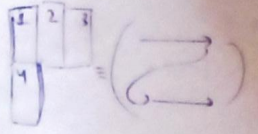
Here 'auto' → adds responsiveness
(each row will fit to 100% of horizontal available space. (grid-(r--)-(r--))
each block/column will only take min space to write (grid-(r--)-(r--))

∴ rows → 100px auto ; # fit content
∴ columns → 200px auto ; # tries to go 100%.

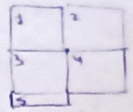
3) ex: grid-(c--)-(c--) : 200px minmax(400px, 800px) ;

minmax(400px, 800px) ~ space available = 800px, when space ↓ = 400px (min.)!

4) grid-(r--)-(r--) : repeat (2, 200px) ;
grid-(c--)-(c--) repeat (3, 100px) ;
{if we only have 4 divs inside the above class}



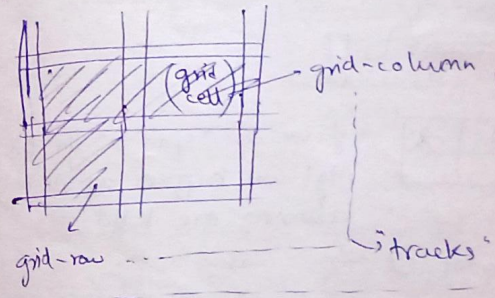
Now suppose we have 5 divs &
(r--)-(r--)-(r--) : 200px, 200px ;
" " (c--) 200px, 200px ;



(we get customatic size based on its height & existing column size)

5) (r--)-(r--)-(r--) :
(r--)-(r--)-(r--) ;
grid-auto-rows : 300px ;
↳ if any 'div' gets added in future, then it will get this property.

- Grid Placement

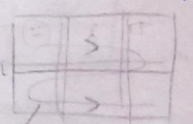


exercise:

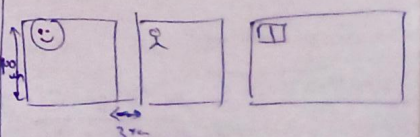
```
<div class="container">
  <div class="item cowboy">☺</div>
  <div class="item ...">♀</div>
  <div class="item book">🕮</div>
</div>
```

CSS

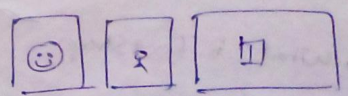
```
.container{
  height : 100vh;
  display : grid ;
  gap : 3rem ;
  grid-template-columns : 1fr 1fr 1fr ;
  grid-template-rows : 1fr 1fr ;
}
.item{
  font-size : 5 rem ;
  color : white ;
  font-family : Arial ;
  background-color : blueviolet
}
```



Preview:



goal

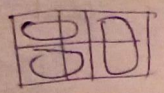


Using flexbox

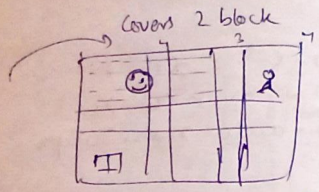
Sol:

```
.item {
    display: flex;
    justify-content: center;   → along main axis of flexbox (here x-axis)
    align-items: center;       → along y-axis
}
```

If we need:

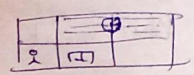


If we do: grid-column: span 2



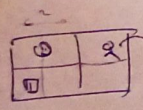
(shortcut)

```
grid-column-start : 2;
grid-column-end   : -1;
```



Similarly we have: grid-row

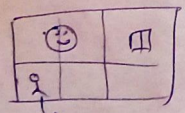
Now our image.



→ if we do "span 2" for this, ntg! will happen as no more columns are left.

Now (in gen., all ele have order = 0)

```
.astro {
    order: 1;
}
```



order 0

order 1

it goes below.

* color: #E58331 80 → transparency.

BOOTSTRAP

- What is BootStrap

* go to : bootstrap.com → copy link & copy inside <head> </head>

Bootstrap: pre-defined sets of CSS

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

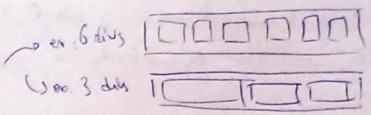
(if we want to overwrite bootstrap, then we include css ref. next line.)

* align-items, justify-content (mention sometimes, to make effective styling)

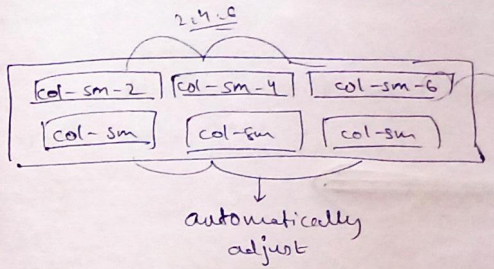
height: 100vh → (covers full webpage)

Bootstrap Layout

Bootstrap is responsive



→ equal sized divs



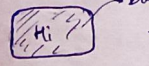
→ screen divided to 12 parts

sm → 6 means 6/12

col-

- sm – small → mobile
- md – medium → ipad
- lg – large → laptop
- xl – xtralarge → desktop breakpoint
- xxl – ... large → TV

Bootstrap Components (refer bootstrap website)

- 

→ for buttons: <button type="button" class="btn btn-primary">Hi</button>

imp. → types:
- primary
- secondary
- success
- Danger
- warning
- info
- light
- dark

→ Check "cards"

"navbars"

→ we can copy svg 'file' to get icons

* dark mode: <html lang="en" data-bs-theme="dark">

my → margin in y-axis
mt → margin top
mx → margin in x-axis
mb → margin bottom

WEB DESIGN SCHOOL

Understanding color Theory

Red = love, energy, intensity ; green = freshness, safety, growth
Yellow = joy, intellect, attention ; blue = stability, trust, serenity
purple = royalty, wealth, femininity

→

INTRODUCTION TO JAVASCRIPT

In Chrome Developer Tools;

⇒ alert("Hi"); → pop up notification saying "Hi"

⇒ prompt("Your Name"); → pop up notification saying "Your Name" & we can type there!

keyword → variable

⇒ var x = "Ankit";

⇒ Variable naming rules in js = same in cpp

⇒ var name = "Angela"
name.slice(0,3) } → Ang
name.toUpperCase(); → ANGELA
name.toLowerCase(); → angela
name.length; → 6

Creating the func (→ can leave empty if also)
⇒ function getMilk(bottles); (→ variable name)

Calling the func
getMilk(2);

⇒ x = 3.3333
alert("Round is" + Math.floor(x)); (Math.floor → round of to lowest val)

⇒ x^2 = Math.pow(x,2);

⇒ var n = Math.random(); (generates random no. b/w [0-1))

If we want to get dice no.s
n = n*6 : [0,5]
n = Math.floor(n) + 1 : [1,6]

Similarly → Math.round(x) { 3.4 → 3 ; 3.6 → 4 }

(for equality)

```
⇒ if(x === 5){
    alert("Hi");
  }else{
    alert("Bye");
  }
```

] extra condn. → else if( ){ }

⇒ •(===) is equal to
•(!==) is not equal to
•(>) •(<) •(>=) •(<=)
•(&&) and •(||) or
•(!) not

```
⇒ var a = 1;    if(a==b) console.log("yes");  → yes
   var b = "1";  if(a===b) console.log("no");  → no
```

(a==b → doesn't checks datatypes; a===b → checks datatypes too; → prints)

⇒ while (condn){
(code)
}] Similarly for loop

(size of array not mentioned)
⇒ var hi = ["An", "ki", "tv"]

• hi.includes("An") (checks if ("") present in 'hi' or not)

• .push() (adds ele. at end of array)

• How to write JS in code:

method-1:
→ <script type="text/javascript">

</script>

method-2:
→ <script src="index.js"></script>
(in new file → all 'js' written)

Code is preffered to be written at end of code!

button

Properties: .innerHTML, .style, .firstChild
Methods: .click(), .appendclick(), .setAttribute()

(object)

→ document.querySelector("h1").innerHTML = "Bye";

→ document.getElementsByTagName("li"); ⇒ returns array of 'li'

→ document.getElementsByClassName("btn"); ⇒ "

→ document.getElementById("title"); ⇒ returns that line
(or)
document.querySelector("#title")

→ document.querySelector("li a");
↳ means inside li → check 'a' & return it

→ document.querySelectorAll("#list .item");
↳ inside id → list we return all class → "item" in form of array

→ document.querySelectorAll("#list .item")[2].style.color = "blue";

→ document.querySelector("button").classList.add(" ");
.remove(" ");
↳ class name whose feature we can on/off

ex: <h1> Hi </h1>
→ document.querySelector("h1").innerHTML = "Hi"
.textContent = Hi
dqs :*

→ dqS("a").attributes ⇒ returns all attributes that has 'a'
→ dqS("a").getAttribute ⇒ "Hi — google — " } returns output link
→ dqS("a").setAttribute("href", " — bing — ") } → changes link from google to bing!
which attribute we are refering
link of that attribute

→ <button class = "w drum">w</—>
< " " = "a drum">a </—>] → js:
for(var i=0; i< dqSA(".drum").length; i++){
dqSA("drum")[i].addEventListener("click", handleClick);
adds sounds
Keyword
func

function handleClick(){
var audio = new Audio(" ");
audio.play();
this.style.color = "White";
}
so whenever we click on button, it's text changes to white

→ var a = {
hi : 12
name : "AV" }
// for accessing it;
console.log(this.a.hi);

Similarly

→ function Hk(yoE, name, report){
this.yoE = yoE
this.name = name
this.report = report
}
→ predefined var or while passing must be given!
ex: var hk1 = new Hk(9, "AV", "Done");

→ .class-name {
background-image: url(" ");
}
→ uploads image!

→ function buttonAnimation(currentkey){ → adding sound to Drum kit
var activeButton = document.querySelector("." + currentkey);
activeButton.classList.add("pressed");
setTimeout(function(){
activeButton.classList.remove("pressed");
}, 100);
}

refer whole code for better understanding in laptop

JQUERY

→ dqS (or) dqSA = $ → property → value to be changed
→ ex: $("h1").css("color", "red");
→ $("h1").addClass("title"); adds class title. If we want to add another class → give space & write class name inside same "
→ $("h1").removeClass("title"); → removes class
→ $("h1").hasClass("title"); → checks class is present or not
→ $("h1").html("Hey") → converts Hey into italics & displays
" .text("Hey") → displays whole!

For attributes
$("a").attr("href"); → if there → it will return
$("a").attr("href", "—link—"); → if not there, it will set!

Now classes are also attributes
ex: $("h1").attr("class"); → prints all classes modifying h1!

→ Now if we want to add an event listener;

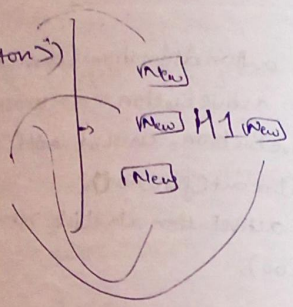
```
$("body").keypress(function(event){
    $("h1").text(event.key);
});
```
→ h1 changes to button we pressed anywhere on body of website

```
$("h1").on("mouseover",function(){
    $("h1").css("color","purple");
});
```
→ when we mouse over h1 color changes to purple

Adding & removing elements;

ex: `$("h1").before("<button>New</button>");`

- .after
- .prepend
- .append



ex: `$("button").remove();` → H1 ↳ all buttons removed!

→
```
$("button").click(function(){
    $("h1").hide();
});
```
similarly: .show(), .toggle()

↳ we click on button & then 'h1' will hide/show/toggle!

→
```
$("button").click(function(){
    $("h1").fadeOut();
});
```
similarly: .fadeIn(), .fadeToggle(), .slideUp(), .slideDown(), .slideToggle()

NODE JS

•"File System" is the native node module that allows us to access the local storage.

ex: `const fs = require("fs");` (mod. that we need has to be entered as string)

```
fs.writeFile("message.txt","Hi",(err)=>{
    if(err) throw err;
    console.log("Yeah!");
});
```
file, message, if error detected

→ Now file created automatically

Syntax:
→ fs.writeFile(file,data[,options],callback)
→ fs.readFile(path[,options],callback)

need to mention, else we get in encoded form

```
fs.readFile('message.txt','utf8',(err,data)=>{
    if(err) throw err;
    console.log(data);
});
```

• To run file in terminal → cd "<file path>" node <filename>

• Creating a 'json' file → "npm init" → fill details

Now installing a package in node → "npm i <package_name>" ex: sillyname

Now → index.js:

```
var generateName = require("sillyname");
var sillyName = generateName();
console.log(`My name is ${sillyName}.`);
```
string interpolation

→ Now instead of using "require" keyword always;→ Go to package.json "type":"module", change

Now we can write, another package

```
import superheroes from "superheroes"
const name = superheroes.random();
console.log(`I am ${name}`);
```

EXPRESS.JS + NODE.JS

- Creating an express server:
  1) Create directory
  2) Create index.js file
  3) Initialise NPM ⟵ (npm init -y)
  4) Install the express package ⟵ (npm i express)
  5) Write Server application in index.js
  6) Start server

```
import express from "express";
const app = express();
const port = 3000;   → port
app.listen(port, () => {      → call back func.
    console.log(`Server running at ${port}`);
});
```

- netstat -ano | findstr "LISTENING" → can see which ports are open

app.listen() → starts the server & makes it listen on a specified port

↳ running (node index.js) ⟶ Type in chrome "localhost:3000" → shows 'Cannot GET /'

(HTTP: HyperText Transfer Protocol)

During HTTP requests; Requests Vocab:

- GET → request resource
- POST → sending resource
- PUT → replace resource
- PATCH → patch up a resource
- DELETE → deletes resources

(PUT, PATCH: update)

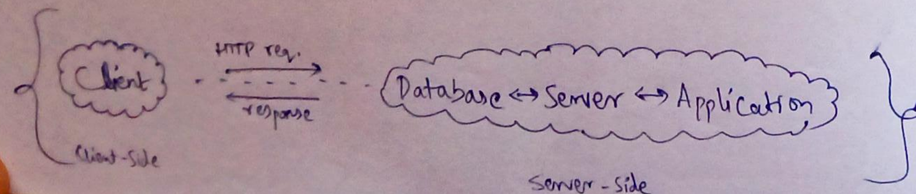
Add:

```
app.get("/", (req, res) => {      ← root of homepage
    res.send("Got it");
});
```

↳ prints 'Got it' → we can also send web-dev, like "<h1> Hi </h1>"

⇒ npm i -g nodemon } installs nodemon (-g: global)

nodemon index.js (filename) } → automatically restarts server if any change is done



- Middlewares → come b/w client & server

ex:

```
import express from "express";
import {dirname} from "path";
import {fileURLToPath} from "url";
import bodyParser from "body-parser";      → Middleware!
const __dirname = dirname(fileURLToPath(import.meta.url));

app.use(bodyParser.urlencoded({extended: true}));      → html form, so using!

app.post("/submit", (req, res) => {      → on clicking submit
    console.log(req.body);      ← it will!
});

app.get("/", (req, res) => {
    res.sendFile(__dirname + "/public/index.html");      → gives path
});      → sends file!
```

In JS → setTimeout (__ , __)

- first argument → (to print) — can also refer
- second argument → after how much delay (in ms)

ex:

```
setTimeout(function(){
    console.log("Hi")
}, 1000)
```

Library:
- collection of func.'s & methods that you can call to achieve a certain task.
- It's like a playground. When you go to a playground you can choose what games you want to play, where you want to play, & how you want to play them. You can slide or climb. You are in control of what you do.

(ex: Javascript, Numpy)

{OR}

- It's a collection of pre-written code that developers can use to perform specific tasks.
- It provides a set of helper func.'s / objects / modules which your application code calls for specific functionality.
- When you use a library, you're in charge of the flow of application. You are choosing when & where to call the library.

Framework:
- It's a collection of tools & rules that may be used to construct software.
- It gives a structure for organizing & constructing larger programs.
- Framework, on the other hand has defined objects or unimplemented func.'s which the user writes to create a custom application.
- You can say that framework is itself an application.
- When you use a framework, the framework is in charge of the flow.

(ex: Node.js, Django)

* Key difference is "Inversion of Control" aka IoC.

How does JS execute code + call stack:

Javascript Execution Context: {:} → memory creation phase → execution phase
- Global execution context
- Functional execution context
- Eval execution context

→ JS creates global execution context
→ Next it creates memory phase

{ in this phase, the variables are set to undefined until the execution phase (next phase) & the func.'s are set to their defn's. }

→ Next it creates execution phase.

→ Execution phase - In this phase, the variables are initialised to given values & when the func.'s are called, it creates a memory phase & execution phase for the func. just like the main one.
→ Returned value is passed to global environment & execution context gets deleted as well.

Execution context → execute 1 line of code at a time → each operation waits for last one to complete before executing → by default

Javascript → Synchronous, Single thread

Blocking Code (block the flow of programs) → read file sync

vs

Non-Blocking Code (doesn't block execution) → read file async

REACT

Terminal: {do these first}
> npx ~~create-react-app~~ (node package executor) (software/utility)

; to start → npm run start

Vite → npm create vite@latest → (go to that folder) → npm i → npm run dev

App.jsx → basically a function (always capital) → to bring → its syntax: <App /> {export default App name}

Const ankit = "wassup"
<h1> Hi {ankit}</h1> → Hi wassup

allows to use variables

this is 'evaluated expression'

App.jsx → returns only 1 value, so in order to return multiple values, we keep it inside 'div'

(ex: <> ... </> ~ (div))

- disable 'red line' for 'unused variables'
  - ↳ /* eslint-disable no-unused-vars */

- TailWind CSS
  - ↳ use closing tag in 'image'

(DEVUI.IO)
↓
(components for website)

useSelector → is a hook for 'react-redux' that allows you to extract data for your redux store state.

useCallback → is a React Hook that lets you cache a function definition between re-renders.
↳ { const cachedFn = useCallback(fn, dependencies) }

useState → is a react hook that lets you add a state variable to your component
↳ { const [state, setState] = useState(initialState) }

useEffect → is a React Hook that lets you synchronize a component with an external system.
↳ { useEffect(setup, dependencies) }

useRef → is a React Hook that lets you reference a value that's not needed for rendering
↳ { const ref = useRef(initialValue) }

useId → is a React Hook for generating unique IDs that can be passed to accessibility attributes.
↳ { const id = useId() }

useContext → is a React Hook that lets you read & subscribe to 'context' from your component
↳ { const value = useContext(SomeContext) }

Actual proj

- npm create vite@latest
- npm i
- npm i @reduxjs/toolkit react-redux react-router-dom appwrite @tinymce/tinymce-react html-react-parser react-hook-form

Using vite & appwrite → we use env. file
↳ import.meta.env.VITE_

Chai ≤ℓ Backend

- Browser ⟷ API ⟷ Backend ⟷ Database

- Computer / Mobile —get→ Server (Express)
  - listen
  - /: home route
  - /login : login setup

{npm init} → ('package.json' file is made)

Professional Approach:

↳ Go to 'package.json' → add a

```
"scripts": {
  "start": "node index.js"
},
```

Now whenever we want file to run, we can type
↳ 'npm run start'

General simple server (index.js)

```
const express = require('express')
const app = express()          ------ [now 'app' got all functionalities of 'express']
const port = 3000

app.get('/', (req, res) => {
    res.send("Hi");
})

app.listen(port, () => {
    console.log(`Port : ${port}`)
})
```

Now few sensitive info. needs to be hidden, so we create '.env' file. → let→ PORT = 3000 (written inside)

Now to access it;
↳ write in index.js ⇝ require('dotenv').config()

```
app.listen(process.env.PORT, () => {

})
```

While adding in github —add→ new file : ".gitignore"
↳ inside this write file names which you don't want to push

```
const port = process.env.PORT || 3000;
```
→ we can also write like this

'Proxy' → use to connect backend → frontend) → helpful (→ react
↳ we add it in: vite.config.js
↳ used to standardize & simply URL requests

- Connect backend with frontend using fullstack proxy & CORS

← Data Modelling for backend with mongoose →

Register
- username
- email
- password
- submit

→ adding any new field will change whole code, so always prepare the 'datapoints' first, what data to store

→ Login page only validates the data in database!

JS :

Math.abs(ele) → absolute value of ele.

map: → basically, it changes the array & saves it in new/same array.

```
> function add10(ele){
      return ele+10;
  }
> let arr = [1,-6,-3,8];
> arr = arr.map(add10)
> arr = arr.map((ele) => {
      return ele/10;
  })
> let brr = arr.map(ele => ele*2);
```

filter: → helps in filtering.

```
> arr = arr.filter((ele) => {
      if(ele%2!=0) return true;
      else return false;
  });
  c.l(arr)
> arr = arr.filter(ele => (ele<4));
  c.l(arr)
```

reduce:

```
let arr = [1,9,2,7,9,5,6,2,8]
let x = arr.reduce(function(a,b){
    return a+b;
})
c.l(x)
```

Sort:

```
let arr = [1,-9,-2,7]
arr = arr.sort()   ⇝ o/p: -2,-9,1,7
```
(gives incorrect sorting)

```
arr = arr.sort((a,b) => a-b)
c.l(arr)
```
→ proper sort

let x = document.querySelector("h1"); → selects 1st instance of this

.querySelectorAll("h1") → selects all instances of this

DOM → console, element selection, change HTML & CSS

Event Listeners → agar main kisi element ko click karoon, ya fir hover karoon, ya element se mouse ko bahar karr doon to kuch changes hoga.

{index.html} → <h1 id="ele1">Hi</h1>

ex. let y = document.~~querySel~~getElementById("ele1");

y.addEventListener("click", function(){ → trigger point

y.style.color = "yellow" → what to do after trigger

});

ex: mouseenter, mouseleave

Generate random no.s : let x = Math.floor(Math.random() * 100)

↳ random no.s btw 1 → 100