Chapter 0 - Introduction

HTML → Hyper Text Mark Up Language

HTML is the language of the web. It is used to create websites
We use HTML tags to define look & feel of a website

With understanding of these tags and How to put them together, we can create beautiful websites easily!

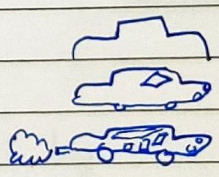
Then Why CSS & JavaScript
HTML is used for defining layout of a page - A barebone page structure

CSS is used to add Styling to that barebone page created using HTML

JavaScript is used to program logic for the page layout eg. What happens when a user hovers on a text, when to hide or show elements etc.

A Beautiful analogy

HTML = Car body (only metal)
CSS = Car paint, decoration etc.
JavaScript = Car engine + Interior logic



We will start learning how to build beautiful layouts in this course.

Installing VS Code

We can use any text editor of our choice. Here I am using VS Code because it is light weight, opensource & from Microsoft.

Go to google, type VS code & install it

Note : You can write HTML even in Notepad. Text editors like VS Code just makes these things easier

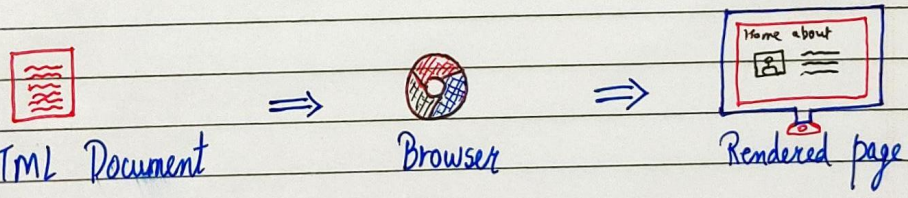
Chapter 1 - Creating our first website

We start building a website by creating a file named index.html index.html is a special filename which is presented when the website root address is typed.

A Basic HTML Page

```
<!Doctype html>  → Specifies this is an html5 doc
<html>  → root of an HTML page
<head>  → Contains page metadata
<title> Harry's Website </title>  → Contains title
</head>
<body>  → The main body of the page (rendered by the browser)
<h1> This is a heading </h1>  → heading tag
<p> My paragraph </p>  → paragraph tag
</body>  → closing body tag
</html>  →
```

A tag is like a container for either content or other HTML tags.



Imp Notes

- Head & body tags are children of HTML tag.
- HTML is the parent of Head & Body tags
- Most of the HTML elements have opening & closing tag with content in between opening & closing tags.
- Some HTML tags have no content. These are called Empty elements eg

- We can either use .htm or .html extension
- You can use "Inspect Element" or "View Page Source" option from chrome to look into a website's HTML code.

HTML element = Start tag + Content + End tag

Comments in HTML

Comments in HTML are used to mark text which should not be parsed. They can help document the source code.

```
<!-- HTML Comment -->
```

Case Sensitivity

HTML is a case insensitive language. <H1> and <h1> tags are the same.

Chapter 1 - Practice Set

1. Inspect your favorite website and change something on the page which is displayed.

2. Go to your favorite website and try to view the page source and write the exact lines of code. Does it clone the website? why?

3. Write any HTML code inside a text file. Does it work if you write it using notepad?

Chapter 2 - Basic HTML Tags

We can add elements inside the body tag to define the page layout.

HTML Element

Everything from starting to the ending tag.

<body> → Opening tag
→ Content ←
</body> → Closing tag

HTML Attributes

Used to add more information corresponding to an HTML tag.

anchor tag
↑
Example : Harry
↓
href attribute

We can either use single or double quotes in attributes

The Heading Tag

Heading tag is used to mark headings in HTML. From h1 to h6, we have tags for the most important to the least least important heading.

<h1> Most Important heading </h1>
<h2> Another heading H2 </h2>
<h3> Another heading H3 </h3>
<h4> Another heading H4 </h4>
<h5> Another heading H5 </h5>
<h6> Another heading H6 </h6>

Note: We should not use HTML headings to make text thick or bold.

The Paragraph Tag

Paragraph tags are used to add paragraphs to an HTML page.

```
<p> This is a paragraph </p>
```

The Anchor Tag

The Anchor tag is used to add links to an existing content inside an HTML page.

```
<a href="https://google.com"> Click me </a>
```

The img Tag

img tag is used to add images in an HTML page

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

↳ relative url of an image

Bold, italic and underline tags

We can use bold, italic and underline tags to highlight the text as follows:

```
<b> This is bold </b>
<i> This is italic </i>
<u> This is underline </u>
```

br tag

The br tag is used to create line breaks in an HTML document.

big and small tags

We can make the text a bit larger and a bit smaller using big and small tags respectively.

hr tag

<hr> tag in HTML is used to create a horizontal ruler often used to separate the content.

Subscript & superscript

We can add subscript and superscripts in HTML as follows:

```
<sub> this </sub> is subscript
<sup> this </sup> is superscript
```

pre tag

HTML always ignores extra spaces and newlines. In order to display a piece of text as is, we use pre tag

```
<pre>
    This    is written
    using   pre
    tag
</pre>
```

⇒ Rendered as-is

Chapter 2 - Practice Set

1. Create an HTML page with a heading (title heading), a primary heading and a sub-heading.
Which tags did you use?

2. Create a page with 5 wallpaper images taken from the internet

3. Use br and hr tags to display a piece of text with linebreaks.

4. Try to write the following chemical equation using HTML.

$$C + O_2 \longrightarrow CO_2$$

5. Try to write a wikipedia article using HTML.

Chapter 3 - Creating a page layout

When we use the right tag in right place, it results in a better page layout, better indexing by search engines and better user experience.
We use the following tag to get the job done

<header> → Contains nav tag
<main>
<footer>
} Website layout

Inside the main tag we insert the following tags:

<main> → The main opening tag
<section> → A page section
<article> → a self contained content
<aside> → Content aside from the content (eg Ads etc.)
</main> → The main closing tag

Creating a page like this is not necessary but it creates a readable & structured layout.
Also they are useful for SEO.

Link attributes
Contact us → Contact page opens in same tab
Contact us
↳ opens in a new tab

We can put any content inside an anchor tag (images, headings etc are all allowed)

If the page is inside a directory, we need to make sure that we link to the correct page.

↳ Same applies to img tag as well

We can add links to images like this

```
<a href="/about"><img src='a.jpg' width="120"></a>
```

↳ Height will be set automatically

The Div tag

div tag is often used as a container for other elements

div is a block level element.

↳ Always takes full width

The Span tag

Span is an inline container.

↳ Takes as much width as necessary

Chapter 3 - Practice Set

1. Create an SEO friendly wesbite using HTML.
2. Create an HTML page which opens google when clicked on an image.
3. Create a website which has your 5 top used websites bookmarked. The links should open in a new tab.

Chapter 4 - Lists, tables & forms

Lists

Lists are used to display content which represents a list

Unordered list: Used to list unordered items

```
<ul>
    <li> Home </li>
    <li> About </li>
    :
</ul>
```

ordered list: used to list ordered items

```
<ol>
    <li> Phone </li>
    <li> PC </li>
    <li> Laptop </li>
</ol>
```

Tables

The <table> tag is used to define tables in HTML.
It is used to format & display tabular data.

tr tag: used to display table row
td tag: used to display table data

th tag: used in place of table data for displaying table headers

We can define as many table rows as we want.

To add a caption to the table, we use <caption> tag inside table.

thead tag: used to wrap table head (caption & tr with th)
tbody tag: used to wrap the table body.

Colspan attribute
This attribute is used to create cells spanning multiple columns.

```
<th colspan="3">Harry</th>
```
↳ Spans 3 columns

HTML forms
An HTML form is used to collect input from the user
form tag is used for the same

```
<form>
-- Element of the form --
</form>
```

There are different form elements for different kinds of user input

- input element: Can be of type text, checkbox, radio, button and submit. We also have a 'file' type
- textarea element: Defines a multi line text input. Cols and rows attributes can be used to size the textarea.
- select element: Defines a drop down list

Note : you dont have to remember all the tags, you will automatically memorize them with practice

Embedding Videos

Video tag is used to play videos in HTML

```
<video src='harry.mp4'> Error </video>
```

Attributes for video

We can use :

- Width : To adjust width of a vido (Height automatically adjusts)
- We can use autoplay/loop to autoplay or loop the video.

Chapter 4 - Practice Set

1. Create an HTML page with video embedded inside it.
2. Replace this video in 1 with a YouTube video.
3. Create an HTML form for a travel website to book a vacation
4. Create a table displaying score of cricket players in a match using HTML.

Chapter 5 - SEO

We will focus only on HTML standpoint of SEO. We will not be looking into keyword building and content optimization aspect of SEO

Types of SEO

- On page SEO → can be done by HTML developers
- Off page SEO

HTML SEO

HTML developers can implement SEO using the following techniques:

1. Set the title very nice & to the point
2. Set the meta description

```
<meta name="description" content="...">
```

3. Set a nice URL slug
4. Set the meta keywords tag.
5. Set the meta author tag.

```
<meta name="author" content="Harry">
```

6. Set a favicon

7. Compress images & other resources

8. Remove unused HTML/CSS & Js files + Compress them

9. Add alt text to images

Introduction to CSS

HTML is just an skeletal layout of a website. We need CSS to design a website, add styles to it and make it look beautiful.

What is CSS

CSS stands for cascading style sheets
CSS is optional but it converts an off looking HTML page into a beautiful & responsive website

Installing VS Code

We will use Microsoft Visual Studio Code as a tool to edit our code. It is very powerful, free and customizable

Why Learn CSS?

CSS is a very demanded skill in the world of web development. If you are successfully able to master CSS, you can customize your website as per your liking.

Your first line of CSS

Create a .css file inside your directory and add it to your HTML. Add the following line to your CSS

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

This will make your page background as red.

HTML Refresher

HTML is a bunch of tags used to lay the structure of a page.

Download HTML notes as part of these notes for a detailed deepdive. If you know basic HTML, continue!

Chapter 1 - Creating our first CSS Website

We will create our first CSS website in this section.

What is DOM?

DOM stands for document object model. When a page is loaded, the browser creates a DOM of the page which is constructed as a tree of Objects.

HTML id and class attributes

When an HTML element is given an id, it serves as a unique identifier for that element

On the other hand, when an HTML element is given a class, it now belongs to that class. More than one elements can belong to a single class but every element must have a unique id (if assigned).

We can add multiple classes to an element like this

unique id

```
<div id="first" class="c1 c2 c3">
  ...
</div>
```

↳ multiple classes followed by spaces

Three ways to add CSS to HTML

There are 3 ways to add CSS to HTML:

1. <style> tag → Adding <style> ... </style> to HTML
2. Inline CSS → Adding CSS using style attribute
3. External CSS → Adding a stylesheet (.css) to HTML using <link> tag.

CSS Selectors

A CSS selector is used to select an HTML element(s) for styling

```
body {                          → Selector
    color : red ;               → Declaration (property : value)
    background : pink ;         → Declaration
}
```

Element selector

It is used to select an element based off the tagname

For example:

```
h2 {
    color: blue;
}
```

id selector

It is used to select an element with a given id

For example:

```
#first {                        → # is used to target by id
    Color: white;
    background: black;
}
```

Class selector

It is used to select an element with a given class

For example:

```
.red {
    background : red;
}
```

Important Notes:

- We can group selectors like this:

```
h1, h2, h3, div {
    color: blue;
}
```

→ h1, h2, h3 and div will be blue

- We can use element.class as a selector like this:

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

→ all paragraphs of class red will get color of red

- * can be used as a universal selector to select all the elements

```
* {
margin : 0;
padding : 0;
}
```

- An inline style will override external and internal styles

Comments in CSS

Comments in CSS is text which is not parse and is thus ignored

Chapter 1 - Practice Set

1. Create a website with a class red div which has a background color of red and color white.
2. Create an element with id head and verify that background color works on it as inline, external as well as using style tag CSS
3. Create a CSS class one and verify that it works on multiple elements.
4. Create multiple CSS classes and verify that all of these work on the same element
5. Have a look at the MDN CSS reference and try to play around with few key-value CSS rules.

Chapter 2 - Colors & Backgrounds

CSS rules are simple key-value pairs with a selector
We can write CSS rules to change color and set backgrounds

The color property

The CSS color property can be used to set the text color inside an element

```
p {
  color: red;
}
```
→ Text color will be changed to red.

Similarly we can set color for different elements

Types of color values

Following are the most commonly used color values in CSS

1. RGB → Specify color using Red, green, blue values eg: rgb(200, 98, 70)
2. HEX Code → Specify color using hex code.
   eg: #ff7180
3. HSL → Specify the color using hsl values
   eg: hsl(8, 90%, 63%)
   ↳ hue, saturation, lightness

The value of the color or background color is provided as any one of these values

Note: We also have an RGBA and an HSLA values for color but they are rarely used by beginners.
A stands for alpha.

The background-color property

The CSS background-color property specifies the background color of a container

For eg:

```
.brown {
    background-color: brown;
}
```

→ Can be other types of colors as well

The background-image property

Used to set an image as the background.

```
body {
    background-image: url("harry.jpg")
}
```

The image is by default repeated in x & y directions

The background-repeat property

Can be any of:

- repeat-x → repeat in horizontal direction
- repeat-y → repeat in vertical direction
- no-repeat → image not repeat

See more possible values at MDN docs

The background-size property

Can be following:

- cover → fits & no empty space remains
- contain → fits & image is fully visible
- auto → Display in original size
- {{width}} → Set width & height will be set automatically

{{width}} {{height}} → set width & height

Note: Always check the MDN docs to dissect a given CSS property. Remember, practice will make you perfect

The background-position property
Sets the starting position of a background image.

```
div1 {
background-position: left top;
}
```

The background-attachment property
Defines a scrollable / non-scrollable character of a background image.

```
div2{
background-attachment: fixed
}
```

The background shorthand
A single property to set multiple background properties.

```
div 3 {
background: red url('img.png') no-repeat fixed right top;
}
```

color ↓ image ↓ repeat

One of the properties can be missing given the others are in order.

↳ {{width}} {{height}} → Set width & height

Note: Always check the MDN docs to dissect a given CSS property. Remember, practice will make you perfect

The background-position property
Sets the starting position of a background image.

```
.div1 {
  background-position: left top;
}
```

The background-attachment property
Defines a scrollable / non-scrollable character of a background image.

```
.div2 {
  background-attachment: fixed
}
```

The background shorthand
A single property to set multiple background properties.

```
.div 3 {
  background: red url('img.png') no-repeat fixed right top;
}
```

color — image — repeat — attachment — position

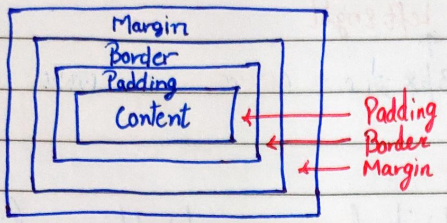
One of the properties can be missing given the others are in order.

Chapter 2 - Practice Set

1. Create a dark blue navigation bar with light color items
2. Change the color of the main container on your page to dark red.
3. Create a div and add a background image with a given width and height
4. Create a vertical box and add a fixed non scrolling background to it
5. Verify that the background shorthand property works with some of the values skipped.

Chapter 3 - CSS Box Model

The CSS box model looks at all the HTML elements as boxes



Setting width & Height

We can set width and height in CSS as follows

```
#box {
    height : 70px;
    width : 70px;
}
```

Note that the total width/height is calculated as follows:

Total height = height + top/bottom padding + top/bottom border + top/bottom margin

Setting Margin & Padding

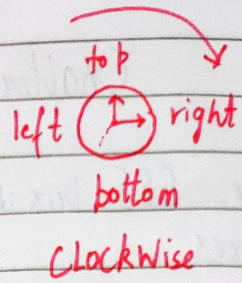
We can set margin and padding as follows:

```
.box {
    margin : 3px;
    padding : 4px;
}
```

Sets top, bottom, left & right values

```
.boxMain {
    margin : 7px 0px 2px 11px;
    }
```

top right bottom left




```
.boxLast {
    margin : 7px 3px;
    }
```

top & bottom Left & right

We can also set individual margins / paddings like this :

```
margin-top : 7opx
margin-bottom : 3px
margin-left : 8px
margin-right : 9px
```

} Same goes with padding

Setting Borders

We can set the border as follows

```
.bx {
    border-width : 2px;
    border-style : solid;
    border-color : red;
    }
```

} or just set border : 2px solid red; (Shorthand)

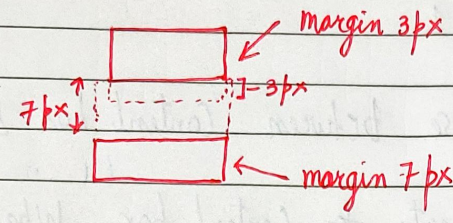
Border Radius

We can set border radius to create rounded borders

```
.div2 {
    border-radius : 7px;
    }
```

Margin Collapse

When two margins from different elements overlap, the equivalent margin is the greater of the two. This is called margin collapse



→ Margin between them is collapsed to the bigger margin

Box Sizing

Determines what out of padding and border is included in elements width and height

Can be content-box or border-box

↳ Include only content in width/height

```
.div1 {
    box-sizing: border-box;
}
```

↳ The content width and height includes content + padding + border

Chapter 3 - Practice Set

1. Create a website layout. Add a header box, 1 content box and one footer.
2. Add borders & margins to 1
3. Did the margin collapse between content box & footer?
4. Add the box-sizing property to content box. What changes did you notice?

Chapter 4 - fonts & display

The display property

The CSS display property is used to determine whether an element is treated as a block/inline element & the layout used for its children.
↳ flexbox/grid/etc.

display: inline

Takes only the space required by the element. No linebreaks before and after. Setting width/height not allowed. (or margin/padding)

display: block

Takes full space available in width and leaves a newline before and after the element

display: inline-block

Similar to inline but setting height, width, margin and padding is allowed. Elements can sit next to each other

display: none vs visibility: hidden

With display: none, the element is removed from the document flow. Its space is not blocked.
With visibility: hidden, the element is hidden but its space is reserved.

text-align property

Used to set the horizontal alignment of a text

```
.div1 {
    text-align: center;
}
```

text-decoration property
Used to decorate the text
Can be overline, line-through, underline, none

text-transform property
Used to specify uppercase and lowercase letters in a text.

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

line-height property
used to specify the space between lines.

```
.Small {
  line-height : 0.7;
   }
```

Font
Font plays a very important role in the look and feel of a website

font-family
font family specifies the font of a text.
Can hold multiple values as a "fallback" system

```
p {
  font-family : "Times new Roman", monospace;
  }
```

↳ Always do this to ensure the correct font of your choice is rendered

Web Safe Fonts

These fonts are universally installed across browsers.

How to add Google fonts

In order to use custom google fonts, go to google fonts then select a style and finally paste it to the style.css of your page.

Other font properties

Some of the other font properties are listed below:

font-size → Sets the size of the font
font-style → Sets the font style
font-variant → Sets whether text is displayed in small-caps
font-weight → Sets the weight of the font

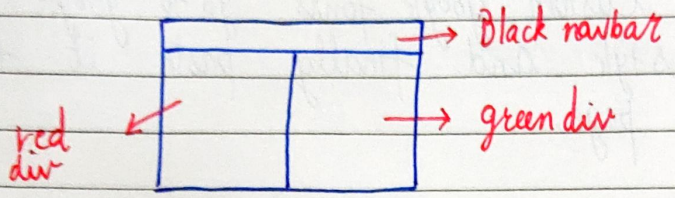
Generic families

Broad class of similar fonts eg. Serif, Sans-Serif
Just like when we say fruit, it can be any fruit.
When we say Serif it can be any Serif font.

font-family → Specific
Generic family → Generic

Chapter 4 - Practice Set

1. Create the following website layout



2. Add a footer with Google font "Ballu Bhai" to (1)

3. Remove the underlines from links in (1)

4. Demonstrate the difference between display: none and visibility: hidden using a div

5. Change the foter to all uppercase in (1)

Chapter 5 - Size, position & Lists

There are more units for describing size other than 'px'
There are rem, em, vw, vh, percentages etc.

Whats wrong with pixels?

Pixels (px) are relative to the viewing device.
For a device with size 1920x1080, 1px is 1 unit out of 1080/1920.

Relative lengths

These units are relative to the other length property.
Following are some of the most commonly used relative lengths

1. em → Unit relative to the parent font size
   ↳ em means "my parent element's font size"
2. rem → Unit relative to the root font size (<Html> tag)
3. vw → Unit relative to 1% viewport width.
4. vh → Unit relative to 1% viewport height.
5. % → Unit relative to the parent element

min/max-height/width property

CSS has a min-height, max-height, min-width and max-width property.
If the content is smaller than the minimum height, minimum height will be applied.

Similar is the case with other related properties

The position property
Used to manipulate the location of an element
Following are the possible values:

- static: The default position. top/bottom/left/right/z-index has no effect.
- relative: The top/bottom/left/right/z-index will now work. Otherwise the element is in the flow of document like static.
- absolute: The Element is removed from the flow & is relatively positioned to its first non-static ancestor. top/bottom etc works
- fixed: Just like absolute except the element is positioned relative to the browser window
- Sticky: The Element is positioned based on user's Scroll position

List-style property
The list style property is a shorthand for type, position & image

```
ul {
    list-style : square inside url('harry.jpg')
}
```

list-style-position

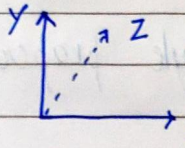
list-style-type

list-style-image

z-index property

The z-index property specifies the stack order of an element.

It defines which layer will be above which in case of overlapping elements



=> Z is the third dimension.

Chapter 5 - Practice Set

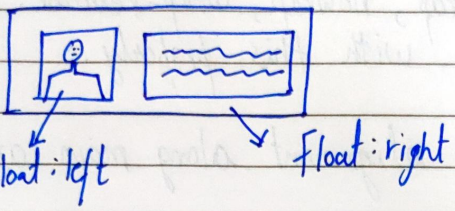
1. Create a responsive navbar using relative lengths
2. Create a sticky navbar using position property
3. Demonstrate the use of list-style property using a UL as example
4. Demonstrate the use of z-index using an example

Chapter 6 - Flexbox

Before we look into the CSS flexbox, we will look into float and clear properties.

The float property

float property is simple. It just flows the element towards left/right



The clear property

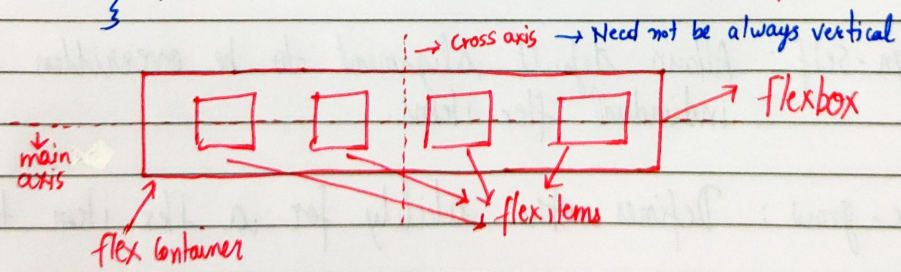
Used to clear the float. It specifies what elements can float beside a given element

The CSS flexbox

Aims at providing a better way to layout, align and distribute space among items in a container.

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

⇒ Initialize a flexbox



flex-direction property
Defines the direction towards which items are laid.
Can be row (default), row-reverse, column, column-reverse

Flex properties for parent (flex container)
following are the properties for the flex parent:

1. flex-wrap: Can be wrap, nowrap, wrap-reverse. Wrap items as needed with this property.
2. justify-content: Defines alignment along main axis.
3. align-items: Defines alignment along cross axis.
4. align-content: Aligns a flex container's lines when there is extra space in the cross axis.

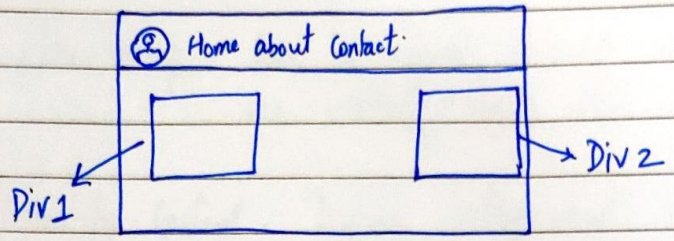
Flex properties for the children (flex items)
following are the properties for the flex children:

1. order: Controls the order in which the items appear in the flex container
2. align-self: Allows default alignment to be overridden for the individual flex items.
3. flex-grow: Defines the ability for a flex item to grow

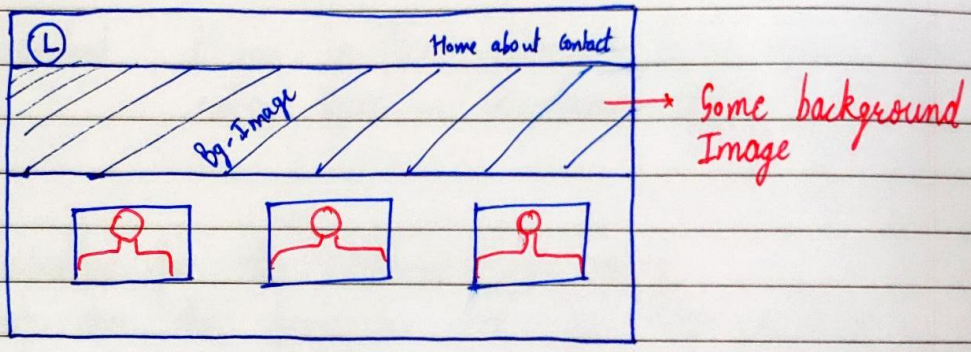
4 flex-shrink : Specifies how much a flex item will shrink relative to the rest of the flex items.

Chapter 6 - Practice Set

1. Create a layout of your choice using float.
2. Create the same layout in (1) using flexbox.
3. Create the following navigation bar using flexbox



4. Create the following layout using flexbox:



Chapter 7 - CSS Grid & Media Queries

A CSS grid can be initialized using :

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

All direct children automatically becomes grid items

The grid-column-gap property
Used to adjust the space between the columns of a CSS grid

The grid-row-gap property
Used to adjust the space between the rows of a CSS grid.

The grid-gap property
Shorthand property for grid-row-gap & grid-column-gap

```
.container {
    display: grid;
    grid-gap: 40px 100px;
}
```

row → 40px
column → 100px

Note : For a single value of grid-gap, both row and column gaps can be set in one value.

Following are the properties for grid container:

1) The grid-template-columns property can be used to specify the width of columns

```
.container {
    display: grid;
    grid-template-columns: 80px 120px auto;
}
```

2) The grid-template-rows property can be used to specify the height of each row

```
.container {
    display: grid;
    grid-template-rows: 70px 150px;
```

3) The justify-content property is used to align the whole grid inside the container.

4) The align-content property is used to vertically align the whole grid inside the container.

Following are the properties for grid item:

1) The grid-column property defines how many columns an item will span.

```
.grid-item {
    grid-column: 1/5;
}
```

2. The grid-row property defines how many rows an item will span.

3. We can make an item to start on column 1 and span 3 columns like this:

```
.item {
    grid-column: 1 / span 3;
    }
```

CSS Media Queries

Used to apply CSS only when a certain condition is true.

Syntax:

```
@media only screen and (max-width: 800px) {
    body {
        background: red;
    }
}
```

Chapter 7 - Practice Set

1. Create a header with content using CSS grid.
2. Create the layouts created in Chapter 6 - Practice Set using CSS grid.
3. Create a webpage which is green on large devices, red on medium & yellow on small devices.

Chapter 8 - Transforms, Transitions & Animations

Transforms are used to rotate, move, skew or scale elements. They are used to create a 3-D effect

The transform property
Used to apply a 2D or 3D transformation to an element

The transform-origin property
Allows to change the position of transformed elements
2D transforms → can change x & y axis
3D transforms → can change Z axis as well

CSS 2D transform methods
You can use the following 2-D transforms in CSS:

1. translate()
2. rotate()
3. scaleX()
4. scaleY()
5. skew()
6. matrix()
7. scale()

CSS 3D transform methods

1. rotateX()
2. rotateY()
3. rotateZ()

CSS Transitions

Used to change property values smoothly, over a given duration.

The transition property

The transition property is used to add transition in CSS.

Following are the properties used for CSS transition.

1. transition-property → The property you want to transition
2. transition-duration → Time for which you want transition to apply
3. transition-timing-function → How you want the property to transition
4. transition-delay → Specifies the delay for the transition

All these properties can be set using a single shorthand property

```
transition : width 3s ease-in 2s;
```

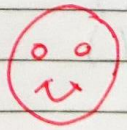
(2) duration, (4) delay

(1) property, (3) timing-function

Transitioning multiple properties

We can transition multiple properties as follows:

```
transition : opacity 1s ease-out 1s, transform 2s ease-in;
```



Yes you can skip transition delay here!

CSS Animations

Used to animate CSS properties with more control.
We can use @keyframes rule to change the animation from a given style to a new style.

```
@keyframes harry {
  from { width: 20px; }  → Can change multiple properties
  to { width: 31px; }
}
```

Properties to add Animations

Following are the properties used to set animation in CSS:

1. animation-name → name of the animation
2. animation-duration → How long does the animation run?
3. animation-timing-function → Determines speed curve of the animation
4. animation-delay → Delay for the start of an animation
5. animation-iteration-count → Number of times an animation shud run
6. animation-direction → Specifies the direction of the animation

The animation shorthand

All the animation properties from 1-6 can be applied like this:

```
animation: harry 6s linear 1s infinite reverse;
           (1)   (2)  (3)  (4)    (5)     (6)
```

Using percentage value states with animation
We can use % values to indicate what should happen when a certain percent of animation is completed

```
@ Keyframes harry {
  0% {
      width: 20px;
     }
  50% {
      width: 80px;
      }
  100% {
      with: 200px;
      }
  }
```

⇒ Can add as many intermediate properties as possible

Chapter 8 - Practice Set

1. Create a thin progress bar animation for a Website
2. Create the same progress bar using transition
3. Create a rotating image animation using CSS
4. Create a slider with 3 images using CSS.

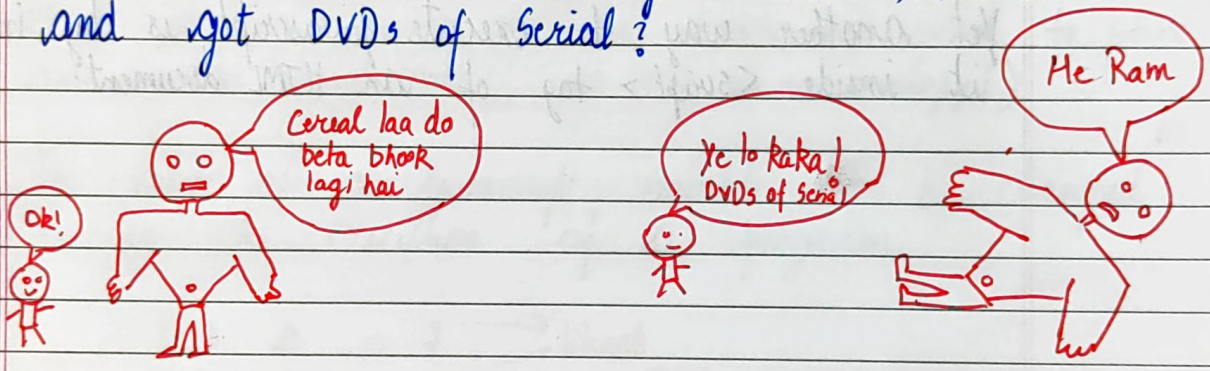
Project 1 - E Commerce Website

Create a homepage for an E-Commerce website.
Use media Queries to make it responsive

Introduction to programming

Programming is a way to talk to computers.
A language like Hindi, English or Bengali can be used to talk to a human but for computers we need straightforward instructions

Computer is Dumb!
When was the last time you ordered some cereal and got DVDs of Serial?



Programming is the act of constructing a program, a set of precise instructions telling a computer what to do.

What is Ecmascript?
ECMAScript is a standard on which Javascript is based!
It was created to ensure that different documents on javascript are actually talking about the same language.

JavaScript & ECMAScript can almost always be used interchangably
JavaScript is very liberal in what it allows.

How to execute JavaScript?

JavaScript can be executed right inside one's browser. You can open the javascript console and start writing javascript there

Another way to execute javascript is a runtime like Node.js which can be installed and used to run javascript code.

Yet another way to execute javascript is by inserting it inside <script> tag of an HTML document.

Chapter 1 - Variables & Data

Just like we follow some rules while speaking english (the grammar), we have some rules to follow while writing a javascript program. The set of these rules is called Syntax in javascript.

What is a Variable?

A Variable is a container that stores a value. This is very similar to the containers used to store rice, water and oats (Treat this as an analogy!)

The value of a javascript variable can be changed during the execution of a program

```
var a = 7;   → literal
let a = 7;   ⇒ Declaring Variables
    ↑   ↑
Identifier  assignment operator
```

Rules for choosing variable names

→ Letters, digits, underscores & $ sign allowed.
→ Must begin with a $, _ or a letter.
→ JavaScript reserved words cannot be used as a variable name
→ Harry & haRRy are different variable (case sensitive)

Var vs let in JavaScript

1. Var is globally scoped while let & const are block scoped
2. Var can be updated & re-declared within its scope
3. let can be updated but not re-declared
4. const can neither be updated nor be re-declared.

5. var variables are initialized with undefined whereas let and const variables are not initialized.
6. const must be initialized during declaration unlike let and var

Primitive Data Types & Objects
Primitive data types are a set of basic data types in javascript
Object is a non primitive datatype in javascript

These are the 7 primitive datatypes in javascript

- Null
- Undefined
- Number
- Boolean
- String
- BigInt
- Symbol

Object
An object in JavaScript can be created as follows.

```
const item = {
    name : "Led Bulb",
    price : "150"
}
```

key ← name, price ; value → "Led Bulb", "150"

Quick Quiz: Write a JavaScript program to store name, phone number and marks of a student using objects.

Chapter 1 - Practice Set

1. Create a variable of type string and try to add a number to it.

2. Use typeof operator to find the datatype of the string in last question

3. Create a const object in javascript. Can you change it to hold a number later?

4. Try to add a new key to the const object in Problem 3. Were you able to do it?

5. Write a JS program to create a word-meaning dictionary of 5 words.

Chapter 2 - Expressions & Conditionals

A fragment of code that produces a value is called an expression. Every value written literally is an expression. for ex: 77 or "Harry"

Operators in JavaScript

1. Arithmetic operators

+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus
++	Increment
--	Decrement

2. Assignment Operators

=	x = y
+=	x = x + y
-=	x = x - y
*=	x = x * y
/=	x = x / y
%=	x = x % y
**=	x = x ** y

3. Comparison Operators

==	equal to
!=	not equal
===	equal value and type
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

4. Logical Operators

&&	logical and
||	logical or
!	logical not

Apart from these, we also have type and bitwise operators. Bitwise operators perform bit by bit operations on numbers.

operands

7 + 8 = 15 → Result

↳ operator

Comments in JavaScript

Sometimes we want our programs to contain a text which is not executed by the JS Engine

Such a text is called comment in Javascript

A comment in Javascript can be written as follows :

```
let a = 2; // this is a single line comment
```
→ Single line Comment

```
/*
  I am a
multiline comment
*/
```
} Multiline comment

Sometimes comments are used to prevent the execution of some lines of code

```
let switch = true;
// switch = false
```
→ Commented line wont execute

Conditional Statements

Sometimes we might have to execute a block of code based off some condition.
For example a prompt might ask for the age of the user and if its greater than 18, display a special message.

In JavaScript we have three forms of if ... else statement.

1. if statement
2. if ... else statement
3. if ... else if ... else statement

If statement

The if statement in JavaScript looks like this:

```
if (condition) {
    // execute this code
    }
```

The if statement evaluates the condition inside the ( ). If the condition is evaluated to true, the code inside the body of if is executed else the code is not executed.

if- else statement

The if statement can have an optional else clause. The syntax looks something like this

```
if (condition) {
    // block of code if condition true
    }
else {
    // block of code if condition false
    }
```

If the condition is true, code inside if is executed else code inside else block is executed

if - else if statement

Sometimes we might want to keep rechecking a set of conditions one by one until one matches. We use if else if for achieving this.

Syntax of if...else if looks like this

```
if (age > 0) {
    console.log("A valid age");
}
else if (age > 10 && age < 15) {
    console.log("but you are a kid");
}
else if (age > 18) {
    console.log("not a kid");
}
else {
    console.log("Invalid Age")
}
```

JavaScript ternary Operator
Evaluates a condition and executes a block of code based on the condition

Condition ? exp1 : exp2

Example Syntax of ternary operator looks like this:

(marks > 10) ? 'yes' : 'No'

↳ if marks are greater than 10, you are passed else not

Chapter 2 - Practice Set

1. Use logical operators to find whether the age of a person lies between 10 and 20?

2. Demonstrate the use of switch case statements in JavaScript

3. Write a Java Script program to find whether a number is Divisible by 2 and 3.

4. Write a JavaScript program to find whether a number is Divisible by either 2 or 3.

6. Print "You can Drive" or "You cannot Drive" based on age being greater than 18 using ternary operator.

Chapter 3 - Loops & functions

We use loops to perform repeated actions. For example - If you are assigned a task of printing numbers from 1 to 100, it will be very hectic to do it manually, Loops help us automate such tasks

Types of loops in JavaScript

for loop → loop a block of code no of times
for in loop → loops through the keys of an Object
for of loop → loops through the values of an object
while loop → loops a block based on a specific condition
do-while loop → while loop variant which runs atleast once

The for loop

The syntax of a for loop looks something like this

```
for ( statement 1 ; statement 2; statement 3) {
  // code to be executed
}
```

→ statement 1 is executed one time
→ Statement 2 is the condition base on which the loop runs (loop body is executed)
→ statement 3 is executed everytime the loop body is executed

Quick Quiz : Write a sample for loop of your choice.

The for-in loop
The syntax of for-in loop looks like this

```
for (key in object) {
  // code to be executed
}
```

Quick Quiz: Write a sample program demonstrating for-in loop

Note - for-in loops also work with arrays which will be discussed in the later videos.

The for-of loop
The syntax of for-of loop looks like this

```
for (variable of iterable) {
  // code
}
```

↳ For every iteration
↳ Iterable data structure like Arrays, Strings etc.

Quick Quiz: Write a sample program demonstrating for-of loops

The while loop
The syntax of while loop looks like this:

```
while (condition) {
  // code to be executed
}
```

Note: If the condition never becomes false, the loop will never end and this might crash the runtime.

Quick Quiz: Write a sample program demonstrating while loop.

The do-while loop

The do while loop's syntax looks like this:

```
do {
  // code to be executed  => Executed at least once
}
while (condition)
```

Quick Quiz: Write a sample program demonstrating do while loop

Functions in Java Script

A JavaScript function is a block of code designed to perform a particular task.

Syntax of a function looks something like this:

```
function myFunc () {
    // code
}

function bindFunc (parameter 1, parameter 2) {
    // code
}
```

↳ Function with parameters

↓ Here the parameters behave as local variables

binod func (7, 8) ⟹ Function Invocation

Function invocation is a way to use the code inside the function

A function can also return a value. The value is "returned" back to the caller

```
Const sum =(a,b) => {
                      let c = a+b;
                      return c;
                    }
```

↓ Another way to create & use the function ↑

⟹ Returns the sum

```
let y = sum (1, 3)
console.log(y)
```

→ Prints 4

Chapter 3 - Practice Set

1. Write a program to print the marks of a student in an object using for loop

obj = { harry : 98, rohan : 70, aakash : 7 }

2. Write the program in Q 1 using for in loop

3. Write a program to print "try again" until the user enters the correct number.

4. Write a function to find mean of 5 numbers.

Chapter 4 - Strings

Strings are used to store and manipulate text.
String can be created using the following syntax:

```
let name = "Harry" → Creates a string
name.length
    ↳ This property prints length of the string
```

Strings can also be created using single quotes

```
let name = 'Harry'
```

Template Literals
Template literals use backtics instead of quotes to define a string

```
let name = `Harry`
```

With template literals, it is possible to use both single as well as double quotes inside a string

```
let Sentence = `The name "is" Harry's`
               ↓ backtic   ↓ double quote   ↓ Single quote
```

We can insert variable directly in template literal. This is called string interpolation

```
let a = `This is ${name}` → Prints 'This is a Harry'
                 name is a variable
```

Escape Sequence Characters

If you try to print the following string, JavaScript will misunderstand it

```
let name = 'Adam D'Angelo'
```

We can use single quote escape sequence to solve the problem

```
let name = 'Adam D\'Angelo'
```

Similarly we can use \" inside a string with double quotes

Other escape sequence characters are as follows

- \n → Newline
- \t → Tab
- \r → Carriage Return

String properties and Methods

1. `let name = "Harry"`
   `name.length` → prints 5

2. `let name = "Harry"`
   `name.toUpperCase()` → prints HARRY

3. `let name = "Harry"`
   `name.toLowerCase()` → prints harry

4) let name = "Harry"
(0, 1, 2, 3, 4)
name.slice(2, 4) → prints rr
(from 2 to 4, 4 not included)

5) let name = "Harry"
name.slice(2) → prints rry
(from 2 to end)

6) let name = "Harry Bhai"
let newName = name.replace("Bhai", "Bhau")

7) let name1 = "Harry"
let name2 = "Naman"
let name3 = name1.concat(name2, "Yes")
↳ We can even use + operator

8) let name = " Harry "
let newName = name.trim()
↳ Removes whitespaces

Strings are immutable. Inorder to access the character at an index we use the following syntax

let name = "Harry"
name[0] → Prints H
name[1] → Prints a

Chapter 4 - Practice Set

1. What will the following print in JavaScript?

```
Console.log("har\"".length)
```

2. Explore the includes, startsWith & endsWith functions of a string

3. Write a program to convert a given string to lowercase

4. Extract the amount out of this string

"Please give Rs 1000"

5. Try to change 4th character of a given string. Were you able to do it?

Chapter 5 - Practice Set

1. Create an array of numbers and take input from the user to add numbers to this array.

2. Keep adding numbers to the array in (1) until 0 is added to the array.

3. Filter for numbers divisible by 10 from a given array.

4. Create an array of Square of given numbers.

5. Use reduce to calculate factorial of a given number from an array of first n natural numbers. (n being the number whose factorial needs to be calculated)

Chapter 5 - Arrays

Arrays are variables which can hold more than one value.

```
const fruits = ["banana", "apple", "grapes"]

const a1 = [7, "Harry", false]
```
↳ can be different types

Accessing Values

```
let numbers = [1, 2, 7, 9]

numbers[0] → 1
numbers[1] → 2
```

Finding the length

```
let numbers = [1, 7, 9, 21]
              ↙0  ↓1  ↓2  ↘3

numbers[0] → 1
numbers.length → 4
```

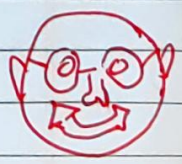
Changing the values

```
let numbers = [7, 2, 40, 9]

numbers[2] = 8
```
↳ "numbers" now becomes [7, 2, 8, 9]

Arrays are mutable

Arrays can be changed



In JavaScript, arrays are objects. The typeof operator on arrays returns object

```
Const n = [1, 7, 9]

typeof n → returns "object"
```

Arrays can hold many values under a single name

Array methods

There are some important array methods in JavaScript. Some of them are as follows:

1. toString() → Converts an array to a string of comma separated values

```
let n = [1, 7, 9]
n.toString() → 1, 7, 9
```

2. join() → joins all the array elements using a separator

```
let n = [7, 9, 13]
n.join("-") → 7-9-13
```

3. pop() → removes last element from the array

```
let n = [1, 2, 4]
n.pop() → updates the original array
          returns the popped value
```

4. push() → Adds a new element at the end of the array

```
let a = [7, 1, 2, 8]
a.push(9) → modifies the original array
          ↳ returns the new array length
```

5. shift() → Removes first element and returns it

6. unshift() → Adds element to the beginning. Returns new array length

7. delete → Array elements can be deleted using the delete operator

```
let d = [7, 8, 9, 10]
delete d[1]        → delete is an operator
```

8. Concat() → Used to join arrays to the given array

```
let a1 = [1, 2, 3]
let a2 = [4, 5, 6]
let a3 = [9, 8, 7]

a1.concat(a2, a3) → Returns [1, 2, 3, 4, 5, 6, 9, 8, 7]
```

↓
Returns a new array
Does not change existing arrays

9> Sort() → sort() method is used to sort an array alphabetically.

```
let a = [7, 9, 8]
a.sort()
```

↳ a changes to [7, 8, 9]
[modifies the original array]

Sort() takes an optional compare function. If this function is provided as the first argument, the sort() function will consider these values (the values returned from the compare function) as the basis of sorting.

10> splice() → splice can be used to add new items to an array

```
Const numbers = [1, 2, 3, 4, 5]
numbers.splice(2, 1, 23, 24)
```

Returns deleted items, modifies the Array
position to add
No of elements to remove
Elements to be added

11> slice() → slices out a piece from an array. It creates a new array

```
Const num = [1, 2, 3, 4]
num.slice(2)    → [3, 4]
num.slice(1, 3) → [2, 3]
```

12. reverse() → Reverses the elements in the source array.

Looping through Arrays
Arrays can be looped through using the classical JavaScript for loop or through some other methods discussed below

1. forEach loop → Calls a function, once for each array element

```
Const a = [1, 2, 3]
a.forEach ((value, index, array) => {
    // function logic
});
```

2. map() → Creates a new array by performing some operation on each array element.

```
Const a = [1, 2, 3]
a.map ((value, index, array) => {
    return value * value;
})
```

3. filter() → Filters ran array with values that passes a test. Creates a new array

```
Const a = [1, 2, 3, 4, 5]
a.filter (greater_than_5)
```

4. reduce method → Reduces an array to a single value

```
Const n = [1, 8, 7, 11]
let sum = numbers.reduce(add)
```

↳ A function

1+8+7+11

5. Array.from → Used to create an array from any other object

```
Array.from("Harry")
```

6. for... of → For-of loop can be used to get the values from an array

7. for ... in → for-in loop can be used to get the keys from an array.

Chapter 6 - JavaScript in the browser

JavaScript was initially created to make web pages alive. JS can be written right in a web page's HTML to make it interactive

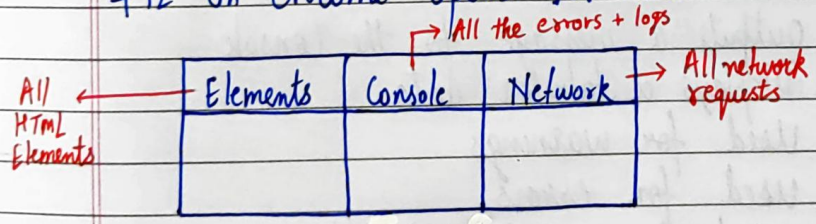
The browser has an embedded engine called the JavaScript engine or the Javascript runtime.

JavaScript's ability in the browser is very limited to protect the user's safety. For example a webpage on http://goofle.com cannot access http://codeswear.com and steal information from there.

Developer tools

Every browser has some developer tools which makes a developer's life a lot easier.

F12 on chrome opens Dev tools



We can also write JavaScript commands in the Console

The script tag

The script tag is used to insert JavaScript into an HTML page

The script tag can be used to insert external or internal scripts

```
<Script>
  alert("Hello")
</Script>
// or...
<Script Src="/js/thisone.js"></Script>
```

The benefit of a separate javascript file is that the browser will download it and store it in its cache

Console object methods
The console object has several methods, log being one of them. Some of them are as follows:

assert() → used to assert a condition
clear() → clears the console
log() → Outputs a message to the console
table() → Displays a tabular data
warn() → Used for warnings
error() → used for errors
info() → Used for special information

You will naturally remember some or all of these with time
Comprehensive list can be looked up on MDN

Interaction : alert, prompt and confirm

alert : Used to invoke a mini window with a msg.

```
alert("hello")
```

prompt : Used to take user input as string

```
inp = prompt("Hi", "No")
```

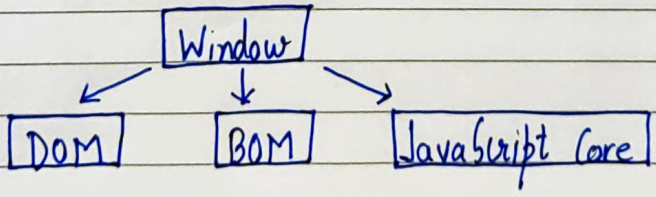
↳ optional default value

Confirm : shows a message and waits for the user to press ok or cancel. Returns true for ok and false for cancel.

The exact location & look is determined by the browser which is a limitation

Window object, BOM & DOM

We have the following when JavaScript runs in a browser



Window object represents browser window and provides methods to control it. It is a global object

classmate
Date
Page

Document Object Model (DOM)

Dom represents the page content as HTML

```
document.body  → Page body as Js object
```

```
document.body.style.background = "green"
```
↳ changes page background to green

Browser Object Model (BOM)

The Browser Object Model (BOM) represents additional objects provided by the browser (host environment) for working with everything except the document.

The functions alert / confirm / prompt are also a part of the BOM

```
location.href = "https://codewithharry.com"
```
↳ Redirect to another URL

Chapter 6 - Practice Set

1. Write a program using prompt function to take input of age as a value from the user and use alert to tell him if he can drive!

2. In Q1 use confirm to ask the user if he wants to see the prompt again

3. In the previous question, use console.error to log the error if the age entered is negative

4. Write a program to change the url to google.com (Redirection) if user enters a number greater than 4

5. Change the background of the page to yellow, red or any other color based on user input through prompt.

Chapter 7 - Walking the DOM

DOM tree refers to the HTML page where all the nodes are objects. There can be 3 main types of nodes in the DOM tree :

1) text nodes
2) element nodes
3) comment nodes

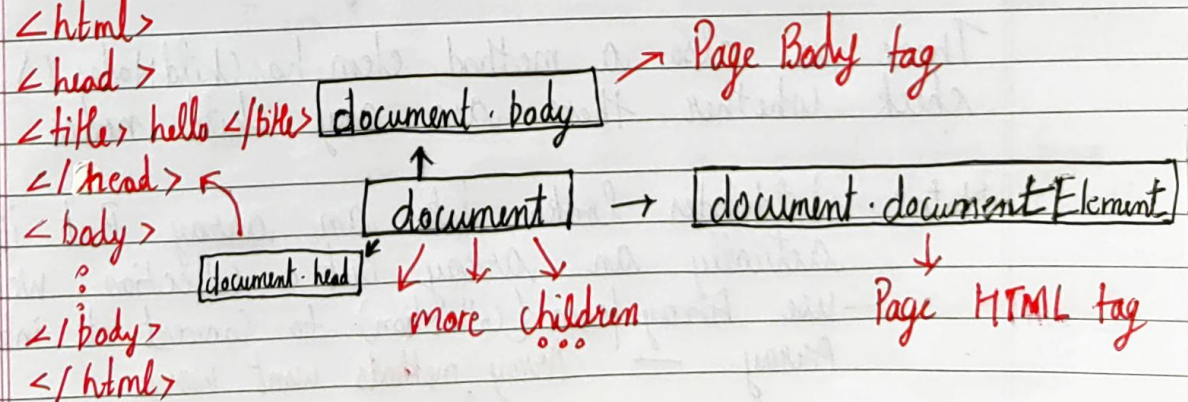
In an HTML page, <html> is at the root and <head> and <body> are its children, etc.

A text node is always a leaf of the tree

Auto Correction

If an erroneous HTML is encountered by the browser, it tends to correct it for example, if we put something after the body, it is automatically moved inside the body. Another example is <table> tag which must contain <tbody>

Walking the DOM



Note : document.body can sometimes be null if the javascript is written before the body tag.

Children of an element
Direct as well as deeply nested elements of an element are called its children

Child nodes → Elements that are direct children
For example head & body are children of <html>

Descendant nodes → All nested elements, children, their children and so on ...

firstChild, lastChild & childNodes

element.firstChild → first child element
element.lastChild → last child element
element.childNodes → All child nodes

Following is always true :

elem.childNodes[0] === elem.firstChild
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild

There is also a method elem.hasChildNodes() to check whether there are any child nodes.

Note : childNodes looks like an array. But its not actually an array but a collection. We can use Array.from(collection) to convert it into an Array. → Array methods went work

Notes on DOM collections

→ They are read-only
→ They are live. elem.childNodes variable (reference) will automatically update if childNodes of elem is changed.
→ They are iterable using for... of loop

Siblings and the parent

Siblings are nodes that are children of the same parent.

→ For example: <head> and <body> are siblings. Siblings have same parent. In the above example its html

→ <body> is said to be the "next" or "right" sibling of <head>, <head> is said to be the "previous" ore "left" sibling of <body>

→ The next sibling is in nextSibling property, and the previous one in previousSibling.
The parent is available as parentNode.

```
alert( document.documentElement.parentNode ); //document
alert( document.documentElement.parentElement ); // null
```

Element only Navigation

Sometimes, we dont want text or comment nodes. Some links only take Element nodes into account. For example

document.previousElementSibling → Previous sibling which is an Element

document.nextElementSibling → next sibling (Element)

document.firstElementChild → first Element child

document.lastElementChild → last Element child.

Table links

Certain DOM elements may provide additional properties specific to their type for convenience.

Table element supports the following properties:

table.rows → collection of tr elements
table.caption → reference to <caption>
table.tHead → reference to <thead>
table.tFoot → reference to <tfoot>
table.tBodies → collection of <tbody> elements
tbody.rows → collection of <tr> inside

tr.cells → collection of td and th
tr.sectionRowIndex → index of tr inside enclosing element
tr.rowIndex → Row number starting from 0

td.cellIndex → no of cells inside enclosing <tr>

Quick Quiz: Print typeof document and typeof window in the console & see what it prints

Searching the DOM

DOM navigation properties are helpful when the elements are close to each other. If they are not close to each other, we have some more methods to search the DOM

→ **document.getElementById**
This method is used to get the element with a given "id" attribute

```
let span = document.getElementById('span')
span.style.color = "red"
```

→ **document.querySelectorAll**
Returns all elements inside an element matching the given CSS selector

→ **document.querySelector**
Returns the first element for the given CSS Selector.
A efficient version of elem.querySelectorAll(css)[0]

→ **document.getElementsByTagName**
Returns elements with the given tag name

→ **document.getElementsByClassName**
Returns elements that have the given CSS class

↳ Dont forget the "s" letter

→ **document.getElementsByName**
Searches elements by the name attribute.

matches, closest & contains methods

There are three important methods to search the DOM

1> elem.matches(css) → To check if element matches the given CSS selector

2> elem.closest(css) → To look for the nearest ancestor that matches the given CSS-selector. The elem itself is also checked

3> elemA.contains(elemB) → Returns true if elemB is inside elemA (a descendant of elemA) or when elemA == elemB

Chapter 7 - Practice Set

1. Create a navbar and change the color of its first element to red.

2. Create a table without tbody. Now use "View page source" button to check whether it has a tbody or not

3. Create an element with 3 children. Now change the color of first and last element to green.

4. Write a javascript code to change background of all <li> tags to cyan.

5. Which of the following is used to look for the farthest ancestor that matches a given CSS selector

(a) matches (b) closest (c) contains (d) none of these

Chapter 8 - Events & other DOM properties

Console.dir function
Console.log shows the element DOM tree
Console.dir shows the element as an object with its properties

tagName / nodeName
Used to read tag name of an element
tagName → only exists for Element nodes
nodeName → defined for any node (text, comment etc.)

innerHTML and outerHTML
The innerHTML property allows to get the HTML inside the element as a string.
↳ Valid for element nodes only

The outerHTML property contains the full HTML. innerHTML + the element itself.

innerHTML is valid only for element nodes. For other node types we can use nodeValue or data.

textContent
Provides access to the text inside the element: only text, minus all tags.

The hidden property
The "hidden" attribute and the DOM property specifies whether the element is visible or not.

```
<div hidden> I am hidden </div>

<div id = "element"> I can be hidden </div>

<script>
element.hidden = true;
</script>
```

Attribute methods

1. elem.hasAttribute(name) → Method to check for existence of an attribute
2. elem.getAttribute(name) → Method used to get the value of an attribute
3. elem.setAttribute(name, value) → Method used to set the value of an attribute.
4. elem.removeAttribute(name) → Method to remove the attribute from elem.
5. elem.attributes → Method to get the collection of all attributes

data-* attributes

We can always create custom attributes but the ones starting with "data-" are reserved for programmers use. They are available in a property named dataset.

If an element has an attribute named "data-one", its available as element.dataset.one

Insertion methods

We looked at some ways to insert elements in the DOM. Here is another way:

```
let div = document.createElement('div') // create
div.className = "alert"                 // set class
div.innerHTML = "<span>hello</span>
document.body.append(div)
```

Here are some more insertion methods:

1. node.append(e) → append at the end of node
2. node.prepend(e) → Insert at the beginning of node
3. node.before(e) → Insert before node
4. node.after(e) → Insert after node
5. node.replaceWith(e) → replaces node with the given node.

Quick Quiz: Try out all these methods with your own webpage.

insert Adjacent HTML / Text / Element

This method is used to insert HTML. The first parameter is a code word, specifying where to insert. Must be one of the following:

1. "beforebegin" - Insert HTML immediately before element
2. "afterbegin" - Insert HTML into element at the beginning
3. "beforeend" - Insert HTML into element at the end.
4. "afterend" - Insert HTML immediately after element.

The second parameter is an HTML String

Example:

```
<div id = "div"> </div>
<script>
  div.insertAdjacentHTML('beforebegin', '<p> Hello </p>');
  div.insertAdjacentHTML('afterend', '<p> Bye </p>');
</script>
```

The output would be:

```
<p> Hello </p>
<div id = "div"> </div>
<p> Bye </p>
```

Node removal

To remove a node, there's a method node.remove()

```
let id1 = document.getElementById("id1")

id1.remove()
```

ClassName and classList

If we assign something to elem.className, it replaces the whole string of classes.

Often we want to add/remove/toggle a single class.

1) elem.classList.add/remove("class") - Adds/removes a class

2) elem.classList.toggle("class") - Adds the class if it doesnt exist, otherwise removes it.

3) elem.classList.contains("class") - Checks for the given class, returns true/false

SetTimeout and setInterval

SetTimeout allows us to run a function once after the interval of time.

Syntax of setTimeout is as follows:

```
let timerId = setTimeout(function, <delay>, <arg1>, <arg2>)
```

timerId ↓ returns a timer Id

<delay> ↓ in ms

clearTimeout is used to cancel the execution (in case we change our mind). For example:

```
let timerId = setTimeout(() => alert("never"), 1000);

clearTimeout(timerId)
```

↳ cancel the execution

setInterval method has a similar syntax as setTimeout:

```
let timerId = setInterval(function, <delay>, <arg1>, <arg2>)
```

All arguments have the same meaning. But unlike setTimeout, it runs the function not only once, but regularly after the given interval of time.

To stop further calls, we can use clearInterval(timerId)

Browser Events

An event is a signal that something has happened. All the DOM nodes generate such signals.

Some important DOM events are:

Mouse events: click, contextmenu (right click), mouseover/mouseout, mousedown/mouseup, mousemove

Keyboard events: keydown and keyup

form element events : submit, focus etc.

Document events : DOMContentLoaded

Handling Events

Events can be handled through HTML attributes

```
<input value = "Hey" onclick = "alert('hey')" type = "button">
```

↳ can be another JS function

Events can also be handled through the onclick property

```
elem.onclick = function() {
                 alert("yes")
                 };
```

Note : Adding a handler with JavaScript overwrites the existing handler

addEventListener and removeEventListener

addEventListener is used to assign multiple handlers to an event.

```
element.addEventListener(event, handler)

element.removeEventListener(event, handler)
```

↳ handler must be the same function object for this to work

The Event Object

When an event happens, the browser creates an event object, puts details into it and passes it as an argument to the handler

```
elem.onclick = function (event) {
    ...
}
```

event.type : Event type
event.currentTarget : Element that handled the event
event.clientX / event.clientY : Coordinates of the cursor

Chapter 8 - Practice Set

1. Write a program to show different alerts when different buttons are clicked

2. Create a website which is capable of storing bookmarks of your favorite websites using href

3. Repeat Q2 using event listeners

4. Write a javascript program to keep fetching contents of a website (Every 5 seconds)

5. Create a glowing bulb effect using classlist toggle method in JavaScript

Chapter 9 - Callbacks, promises & async/await

Asynchronous actions are the actions that we initiate now and they finish later. eg. setTimeout
Synchronous actions are the actions that initiate and finish one-by-one

Callback functions

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete an action.

Here is an example of a callback:

```
function loadScript (src, callback) {
    let script = document.createElement ('script')
    script.src = src
    script.onload = () => callback (script)
    document.head.append (script)
}
```

Now we can do something like this:

```
loadScript ('https://cdn.harry.com', (script) => {
    alert ('script is loaded')
    alert (script.src)
});
```

This is called "callback-based" style of async programming. A function that does something asynchronously should provide a callback argument where we put the function to run after it's complete

Handling errors
We can handle callback errors by supplying error argument like this:

```
function loadScript (src, callback) {
  ...
  ...
  script.onload = () => callback (null, script);
  script.onerror = () => callback (new Error ('failed'));
  ...
}
```

Then inside of loadScript call:

```
loadScript ('cdn/harry', function (error, script) {
  ...
  if (error) {
    // handle error
    }
  else {
    // script loaded
    }
  });
```

Pyramid of Doom

When we have callback inside callbacks, the code gets difficult to manage

```
loadScript((...) {
    loadScript ..
        loadScript ...
            loadScript
                ...
```

← Pyramid of Doom

As calls become more nested, the code becomes deeper and increasingly more difficult to manage, especially if we have real code instead of ...

This is sometimes called "callback hell" or "pyramid of doom"

The "pyramid" of these calls grows towards the right with every asynchronous action. Soon it spirals out of control. So this way of coding isn't very good!

Introduction to Promises

The solution to the callback hell is promises. A promise is a "promise of code execution". The code either executes or fails, in both the cases the subscriber will be notified.

The syntax of a Promise looks like this:

```
let promise = new Promise( function (resolve, reject) {
    // executor
});
```

(predefined in JS engine)

resolve and reject are two callbacks provided by javascript itself. They are called like this:

resolve (value) → If the job is finished successfully
reject (error) → If the job fails

The promise object returned by the new Promise constructor has these properties

1. State: Initially pending, then changes to either "fulfilled" when resolve is called or "rejected" when reject is called

2. result: Initially undefined, then changes to value if resolved (resolve(value)) or error when rejected (reject(error))

Consumers: then & catch
The consuming code can receive the final result of a promise through then & catch

The most fundamental one is then

```
promise.then( function (result) { /* handle */ },
              function (error) { /* handle error */ }
            );
```

If we are interested only in successful completions, we can provide only one function argument to .then():

```
let promise = new Promise (resolve => {
    setTimeout (() => resolve ("done"), 1000);
    });

promise.then (alert);
```

If we are interested only in errors, we can use null as the first argument: .then(null, f) or we can use .catch:

```
promise.catch (alert)
```

promise.finally (() => {}) is used to perform general cleanups

Quick Quiz: Rewrite the loadScript function we wrote in the beginning of this chapter using promises.

Promises Chaining

We can chain promises and make them pass the resolved values to one another like this

```
p.then (function (result) => {          // p is a promise
    alert (result); return 2;
    }).then ...
```

The idea is to pass the result through the chain of .then handlers.

Here is the flow of execution

1. The initial promise resolves in 1 seconds (Assumption)
2. The next .then() handler is then called, which returns a new promise (resolved with 2 value)
3. The next .then() gets the result of previous one and this keeps on going

Every call to .then() returns a new promise whose value is passed to the next one and so on. We can even create custom promises inside .then()

Attaching multiple handlers

We can attach multiple handlers to one promise. They dont pass the result to each other; instead they process it independently.

Let p is a promise

p.then(handler1) →

p.then(handler2) → Runs Independently

p.then(handler3) →

Promise API

There are 6 static methodes of Promise class:

1) Promise.all(promises) → Waits for all promises to resolve and returns the array of their results. If any one fails, it becomes the error & all other results are ignored

2) Promise.allSettled(promises) → Waits for all the promises to settle and returns their results as an array of objects with status and value.

3) Promise.race(promises) → Waits for the first promise to settle and its result/error becomes the outcome.

4) Promise.any(promises) → Waits for the first promise to fulfill (& not rejected), and its result becomes the outcome. Throws AggregateError if all the promises are rejected.

5) Promise.resolve(value) → Makes a resolved promise with the given value.

6) Promise.reject(error) → Makes a rejected promise with the given error.

Quick Quiz : Try all these promise APIs on your custom promises.

Async / Await

There is a special syntax to work with promises in javascript

A function can be made async by using async keyword like this :

```
async function harry(){
    return 7;
}
```

An async function always returns a promise. Other values are wrapped in a promise automatically

We can do something like this :

```
harry().then(alert)
```

So, async ensures that the function returns a promise and wraps non promises in it.

The await keyword

There is another keyword called await that works only inside async functions

```
let value = await promise;
```

The await keyword makes javascript wait until the promise settles and returns its value.

Its just a more elegant syntax of getting the promise result than promise.then + its easier to read & write

Error Handling

We all make mistakes. Also sometimes our script can have errors. Usually a program halts when an error occurs.

The try ... catch syntax allows us to catch errors so that the script instead of dying can do some thing more reasonable

The try ... catch Syntax

The try catch syntax has two main blocks: try and then catch

```
try {
  // try the code

} catch (err) {
  // error handling
  }
```

⇒ The err variable contains an error object

It works like this

1. first the code in try is executed
2. If there is no error, catch is ignored else catch is executed

try catch works synchronously
If an exception happens in scheduled code, like in setTimeout, then try... catch wont catch it :

```
try {
    setTimeout ( function () {
            // error code
        }
catch ...
```

→ Script dies and catch wont work

Thats because the function itself is executed later, when the engine has already left the try ... catch construct.

The error object
For all the built in errors, the error object has two main properties :

```
try {
  hey; // error variable not defined
  } catch (err) {
        alert (err.name)
        alert (err.message)
        alert (err.stack)
    }
```

Throwing Custom Error

We can throw our own error by using the throw syntax

```
if (age > 180) {
    throw new Error("Invalid Age")
    ...
```

We can also throw a particular error by using the built in constructor for standard errors:

```
let error = new SyntaxError(message)
            or
            new ReferenceError(message)
```

The finally clause

The try ... catch construct may have one more code clause: finally

If it exists it runs in all cases:

after try if there were no errors
after catch if there were errors

If there is a return in try, finally is executed just before the control returns to the outer code.

Chapter 9 - Practice Set

1. Write a program to load a JavaScript file in a browser using Promises. Use .then() to display an alert when the load is complete

2. Write the same program from previous question and use async/await Syntax

3. Create a promise which rejects after 3 seconds. use an async/await to get its value. use a try catch to handle its error

4. Write a program using Promise.all() inside an async/await to await 3 promises. Compare its results with the case where we await these promises one by one.

Chapter 11 - Practice Set

1. Create a JavaScript class to create a complex number. Create a constructor to set the real and the complex part

2. Write a method to add two complex numbers in the above class

3. Create a class Student from a class Human. Override a method & see changes

4. See if Student is an instance of Human using instanceof Keyword.

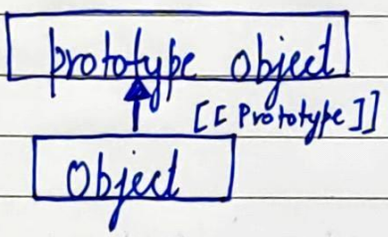
5. Use getters & setters to set and get the real and imaginary parts of the complex number

Chapter 11 - Object Oriented Programming

In programming we often take something and then extend it. for example we might want to create a user object and "admin" and "guest" will be slightly modified variants of it

[[Prototype]]

JavaScript objects have a special property called prototype that is either null or references another object



When we try to read a property from a prototype and its missing, JavaScript automatically takes it from the prototype. This is called "prototypal inheritance"

Setting Prototype

We can set prototype by setting __proto__

Now if we read a property from the object which is not in object and is present in the prototype, JavaScript will take it from prototype

If we have a method in object, it will be called from the object. If its missing in object and present in prototype, its called from the prototype

Classes and Objects

In object-oriented programming, a class is an extensible program-code template for creating objects, providing initial values for state (member variables) and implementation of behavior (member functions)

The basic syntax for writing a class is :

```
Class MyClass {
    // class methods
    constructor() { ... }
    method1() { ... }
    method2() { ... }
    }
```

We can then use new MyClass() to create a new object with all the listed methods

The Constructor method

The Constructor() method is called automatically by new, so we can initialize the object there

Quick Quiz : Create a class user and create a few methods along with a constructor.

Class Inheritance

Class Inheritance is a way for one class to extend another class. This is done by using the extends keyword.

The extends keyword

extends keyword is used to extend another class.

```
class Child extends Parent
```

We can create a class Monkey that inherits from Animal

```
class Monkey extends Animal {
    hide() {
        alert(`${this.name} hides!`);
    }
}

let monkey = new Monkey("Monu")
monkey.run(7); // From Animal
monkey.hide();
```

Method Overriding

If we create our own implementation of run, it will not be taken from the Animal class. This is called Method Overriding

super keyword

When we override a method, we dont want the method of the previous class to go in vain. We can execute it using super keyword.

```
super(a, b)
```

→ call parent constructor

```
run () {
    super. run ( )
    this. hide ( )
    }
```

Overriding Constructor

With a constructor, things are a bit tricky / different According to the specification, if a class extends another class and has no constructor, then the following empty constructor is generated

```
Class Monkey extends Animal {
    // auto generated
    Constructor (... args) {
        super (... args);
    }
}
```

=> Happens if we dont write our own Constructor

Constructors in inheriting classes must call super (...) and do it before using this.

We can also use super. method ( ) in a Child method to call Parent Method

Static method

Static methods are used to implement functions that belong to a class as a whole and not to any particular object

We can assign a static method as follows:

```
Class Employee {
    static sMethod () {
        alert ("Hey");
    }
}
Employee.sMethod()
```

Static methods arent available for individual objects

Getters and Setters
Classes may include getters and setters to get & set the computed properties

Example:

```
class Person {
    ...
    get (name) {
        return this._name;
    }
    set name (newName) {
        this._name = newName;
    }
```

First the name property is changed to _name to avoid the name collision with the getter & setter. Then the getter uses the get keyword as shown above

instanceof Operator

The instanceof operator allows to check whether an object belongs to a certain class

The syntax is :

<obj> instanceof <class>

It returns true if obj belongs to the Class or any other class inheriting from it

Chapter 12 - Advanced JavaScript

There are some JavaScript concepts which make the life of a developer extremely simple. We will discuss some of those in this chapter

IIFE

IIFE is a JavaScript function that runs as soon as it is defined.

```
( function () {
    ...
    ...
  } )();
```

⇒ IIFE Syntax

It is used to avoid polluting the global namespace, execute an async-await, etc.

Destructuring

Destructuring assignment is used to unpack values from an Array, or properties from objects, into distinct variables

```
let [x, y] = [7, 20]
```

x will be assigned 7 and y, 20

```
[10, x, ...rest] = [10, 80, 7, 11, 21, 88]
```

x will be 80 rest will be [7, 11, 21, 88]

Similarly we can destructure objects on the left hand side of the assignment

```
Const obj = { a: 1, b: 2 }
const {a, b} = obj;
```

Some more examples can be found on MDN docs.

Spread Syntax

Spread syntax allows an iterable such as an array or string to be expanded in places where zero or more arguments are expected. In an object literal, the spread syntax enumerates the properties of an object and adds the key-value pairs to the object being created.

Example:

①
```
Const arr = [1, 7, 11]
Const obj = { ...arr }; // { 0: 1, 1: 7, 2: 11 }
```

②
```
Const nums = [1, 2, 7]
Console.log (Sum (...nums)) // 10
```

Other examples can be found on MDN docs

Quick Quiz: Output of the following ??

```
Const a = "the", b = "no"
Const c = { a, b }
Console.log (c)
```

Date
Page

local, global & block scopes

JavaScript has three types of scopes:

1. Block scope
2. Function Scope
3. Global Scope

let & const provide block level scope which means that the variables declared inside a { } cannot be accessed from outside the block

```
{
  let a = 27;
}
// a is not available here
```

Variables declared within a JavaScript function, become local to the function

A variable declared outside a function, becomes global

Hoisting

Hoisting refers to the process whereby the interpreter appears to move the declarations to the top of the code before execution

Variables can thus be referenced before they are declared in JavaScript

hello("Harry")

function hello(name) {
...
}

⇒ Works!

Important Note: JavaScript only hoists declarations, not initializations. The variable will be undefined until the line where its initialized is reached.

Hoisting with let and var
With let and var hoisting is different

console.log(num) → Error if let or const
let num = 6; → with var undefined is printed

function expressions and class expressions are not hoisted

Chapter 12 - Practice Set

1. Write a JavaScript program to print the following after 2 second delay

   Hello
   World

2. Write a JavaScript program to find average of numbers in an array using spread syntax

3. Write a JavaScript function which resolves a Promise after n seconds. The function takes n as the parameter. Use an IIFE to execute the functions with different values of n

4. Write a simple interest calculator using JavaScript.

Exercise 1 - Guess the number

Write a JavaScript program to generate a random number and store it in a variable. The program then takes an input from the user to tell them whether the guess was correct, greater or lesser than the original number.
100 - (no of guesses) is the Score of the user.
The program is expected to terminate once the number is guessed. Number should be between 1 - 100.

Exercise 2 - Snake Water Gun

Use Javascript to create a game of Snake Water & Gun. The game should ask you to enter S, W or G. The computer should be able to randomly generate S, W or G and declare Win or Loss using alert. Use confirm and prompt wherever required

Exercise 3 - Tell me a Joke

elem.innerHTML is used to populate a div with HTML. Search online about this method and create a website with a div tag containing a random joke given an array of jokes. Use Math.random and fetch jokes from the internet (use any website to create the array). Your website should show a random joke on every reload. Min length of your jokes array should be 10.

Exercise 4 - Digital Clock

1. Create a Digital Seconds clock using SetInterval and Date object in JavaScript
The Date object can be used to get the date, time, hours and seconds which can then be updated using setInterval.
Try to keep the UI good looking.

Exercise 5 - Hackerman

Write a javascript program to pretend to look like a hacker. Write an async function which will simply display the following output:

Initializing Hack program ...

Hacking Ashish's username ...

Username found aashish17 ...

Connecting to facebook ...

Try to use HTML & Styling if possible

Exercise 6 - TODO List

Create a TODO List app capable of storing your TODOs in local Storage. Add an option to create, delete and access all the TODOs.

Try to make UI as good as possible

Exercise 7 - Password Generator

Create a JavaScript program capable of generating a password which contains atleast one lowercase, one uppercase and one special characters.
Create a Password class to acheive the same

Exercise 8 - Alarm Clock

The HTML Audio Element Interface can be used to play audio in the browser. Create an alarm clock which displays time and plays sound at a user specified time