LAVAN KUMAR

Web Technology Notes

HTML NOTES

===================

1)WHAT IS WORLD WIDE WEB(WWW)?

world wide web commonly refered as www/w3/web. It is a system interconnected public web pages accessable through internet.

-->If we want to create a web we need atleast 2 components i.e, client computer and a web server.

-->with the help of internet only we are able to connect to the server, without it won't possible.

INTERNET:(Interconnected Network)

=================================

The internet is a global network of interconnected computer networks that use a standardized set of protocols(such as TCP/IP) to facilitate the exchange of data and information.It is a vast infrastructure that connects millions of devices including computers, servers,routers and other network-enable devies across the world.

TCP: Transmission control protocol

IP: Internet protocol

(or)

The internet enables communication information sharing and access to various online sevices and resources. It provides a platform for individuals,organizations and governments to connect,collabraye and exchange information regardless of geographical boundaries.

What is Web Browser?

A web browser is a software application that alllows users to access, view and intract with others on the world wide web.It serves as a user interface for navigating and display web page, websites and web-based applications and also it helps to render the HTML,CSS,Javascript etc.

what is web server?

A web server is a software and hardware that uses http and other protocols to respond to client request made over the www.

http: hyper text transfer protocol

url: uniform resource locator

What is website?

website is the collection of web pages different multimedia content such as text images and viedos which can be accessed by the url,which you can see in the address bar of the browser.

for example: https://www.google.com

How to access websites?

when we type a certain url in a browser search bar, the browser request the page from the webserver and the web server returns the requried web page and it content to the browser.Now it differs how the server returns the requried information in the case of static and dynamic websites.

TYPES OF WEBSITES............

1)static website

2)dynamic website

1)Static Website

web pages are returned by the server which are prebuilt source code files built using simple languages such as html,css,js."There is no processing of the content on the server (according to the user) in static website.web pages are returned by the server with no change."therfore,static websites are fast,also they are less costly.

Note: Static does not mean that it will not respond to user actions,these websites are called static because these cannot be manipulated on the server.

2)Dynamic Website

web pages are returned by the server which are processed during runtime means they are not prebuilt web pages but they are build during runtime according to the user's demand with the help of server-side scripting languages such as php,nodejs,asp.net and many more supported by the server. so they are slower than static websites but updates and intaction with database are possible.

Dynamic websites are used over static websites are updates can be done very easily as compared to static website(Where altring in every page is requried) but in dynamic websites it is possible to do common changes once and it will reflect in all the web pages.

Difference betweeen Static website and Dynamic website?

Static website	Dynamic website:
1)content of web pages can't be change at runtime. 2)It is faster to load as compared to dynamic website 3)cheaper development costs 4)html,css,js is used for developing the websites 5)same content is delevered everytime the page is loaded. examples: any educations websites	1)content of web pages can be changed. 2)It is slower than static website. 3)More development costs. 4)server side languages such as PHP,Nodejs etc are used. 5)content may changes everytime the page is loaded.

installation process for vscode:

step 1:

open google chrome and search "visual studio code for windows" download and install.

step 2:

open vs code and open extensions section and search live server and install.

step 3:

create a folder on desktop and name it as WEB Technology.

step 4:

open that folder and right click and select the open with vs code

or

open that folder in address bar type "cmd" press enter. then it will open command prompt . in command prompt type "code ." press enter.

How to create a html file in vs code

step 1:

in vs code top left side corner we can see one file icon click on it , give your file name with .html file extension. press enter

step 2:

press shit+1 you will get ! and press enter or type html:5 press enter.

what is html tag?

HTML tags are the predefined keywords on a web page that define how your web browser must formate and display your web page.

Almost all tags contains two parts

1.opening tag

2.closing tag

ex: <html></html>

what is an html elements?

The HTML element is everything from the start tag to the end tag

ex: <tagname> content goes here .....</tagname>

what is head tag?

the head element is a container for metadata.(metadat is data about the html document).the head element contains machine readable information about the document like style sheet ,title, script other metadata.

Note:Head tag primarily holds information for machine processing, not for the human readability.

what is body tag?

the body tag defines the document's body. the body element contains all the content of an html document, such as headings,tables,lists etc.

Note: only one body tag should be in a single html document.

HTML Headings

HTML headings are defined with the **<h1> to <h6>** tags.

<h1> defines the most important heading**. <h6>** defines the least important heading.

horizontal line:

The **<hr>** tag in HTML stands for "**horizontal rule**." It is a self-closing tag

that is used to create a thematic break or horizontal line in a webpage.

The <hr> tag is primarily used to separate content or sections within a

document, providing visual clarity and structure to the page.

The <hr> element is used to separate content (or define a change) in an HTML page

Thematic Break: It is often used to denote a thematic break between different sections of content within a webpage. For example, you might use <hr> to separate different topics in an article, distinguish sections of a form, or divide content within a blog post.

The <hr> tag is an empty tag, which means that it has no end tag.

HTML Line Breaks

The HTML <br> element defines a line break.

Use <br> if you want a line break (a new line) without starting a new paragraph.

The <br> tag is an empty tag, which means that it has no end tag.

<p>	Defines a paragraph
<hr>	Defines a horizontal rule in the content
<br>	Inserts a single line break
<pre>	Defines pre-formatted text

HTML Entities

Reserved characters in HTML must be replaced with entities:

< (less than) = &lt;

> (greather than) = &gt;

Formatting tags in html:

<b> - Bold text

<strong> - Important text

<i> - Italic text

<em> - Emphasized text

<mark> - Marked text

<del> - Deleted text

<sub> - Subscript text

<sup> - Superscript text

<abbr> - abbreviation text

<u> - underline the text

LIST IN HTML:

HTML lists allow web developers to group a set of related items in lists.

Types of list in html

1) Ordered HTML list

The HTML <ol> tag defines an ordered list. An ordered list can be numerical or alphabetical.

An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.

The list items will be marked with numbers by default.

The type attribute of the <ol> tag, defines the type of the list item marker

type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman number
type="i"	The list items will be numbered with lowercase roman number

By default, an ordered list will start counting from 1. If you want to start counting from a specified number, you can use the start attribute.

```
example: <ol start="10">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

2)Unordered list

The HTML <ul> tag defines an unordered (bulleted) list.

An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.

The list items will be marked with bullets (small black circles) by default.

The CSS list-style-type property is used to define the style of the list item marker. It can have one of the following values.

disc	Sets the list item marker to a bullet (default)
circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

example:

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

3) Description List:

A description list is a list of terms, with a description of each term.

The <dl> tag defines the description list, the <dt> tag defines the term (name), and the <dd> tag describes each term.

Use the HTML <dl> element to define a description list

Use the HTML <dt> element to define the description term

Use the HTML <dd> element to describe the term in a description list

example:

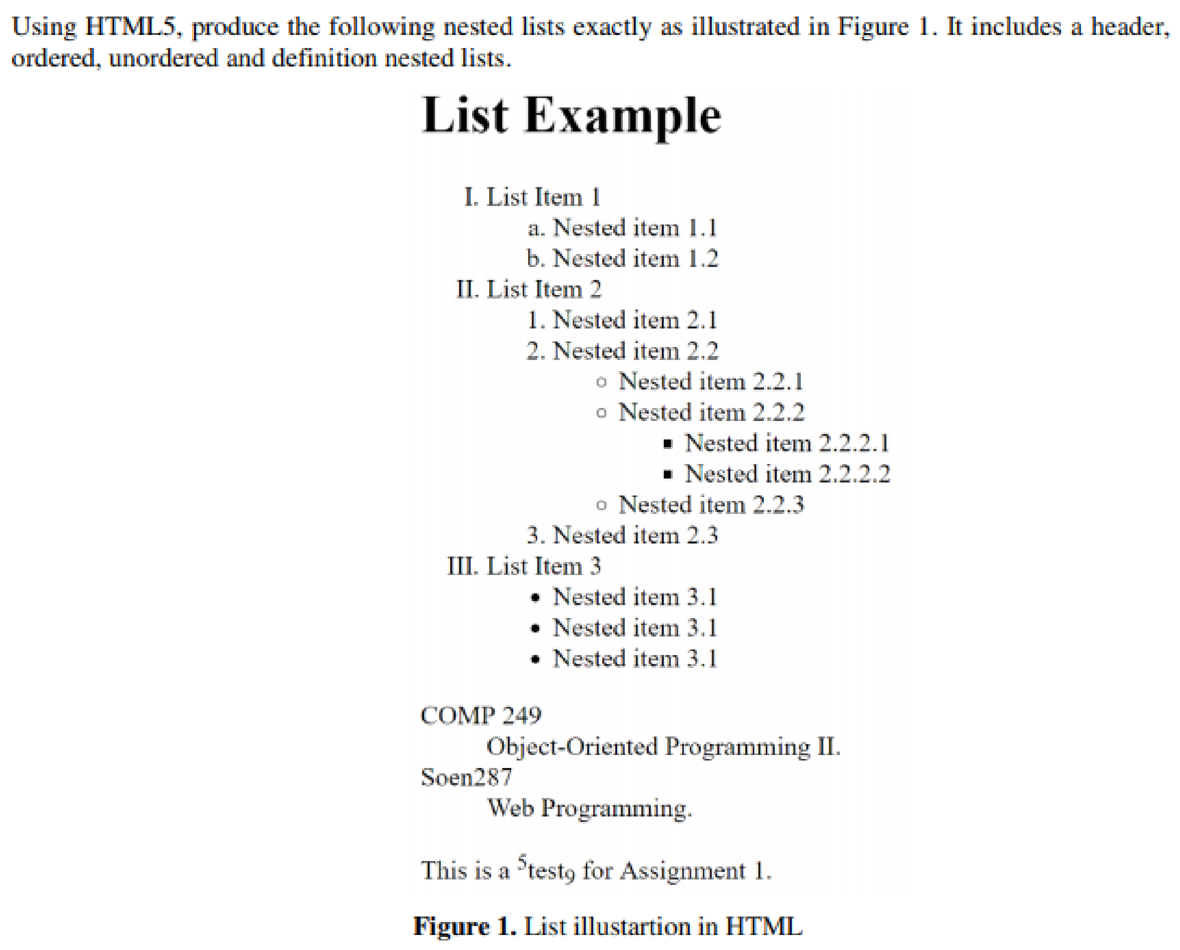
```
<dl>
```

```
  <dt>Coffee</dt>

  <dd>- black hot drink</dd>

  <dt>Milk</dt>

  <dd>- white cold drink</dd>

</dl>
```

4) Nested List

Lists can be nested (list inside list):

example:



Using HTML5, produce the following nested lists exactly as illustrated in Figure 1. It includes a header, ordered, unordered and definition nested lists.

List Example

I. List Item 1
    a. Nested item 1.1
    b. Nested item 1.2
II. List Item 2
    1. Nested item 2.1
    2. Nested item 2.2
        ○ Nested item 2.2.1
        ○ Nested item 2.2.2
            ▪ Nested item 2.2.2.1
            ▪ Nested item 2.2.2.2
        ○ Nested item 2.2.3
    3. Nested item 2.3
III. List Item 3
    • Nested item 3.1
    • Nested item 3.1
    • Nested item 3.1

COMP 249
    Object-Oriented Programming II.
Soen287
    Web Programming.

This is a ^{5}test$_9$ for Assignment 1.

Figure 1. List illustartion in HTML

HTML Tables:

HTML tables allow web developers to arrange data into rows and columns.

`<table></table>` tag is used to defines a table in html.

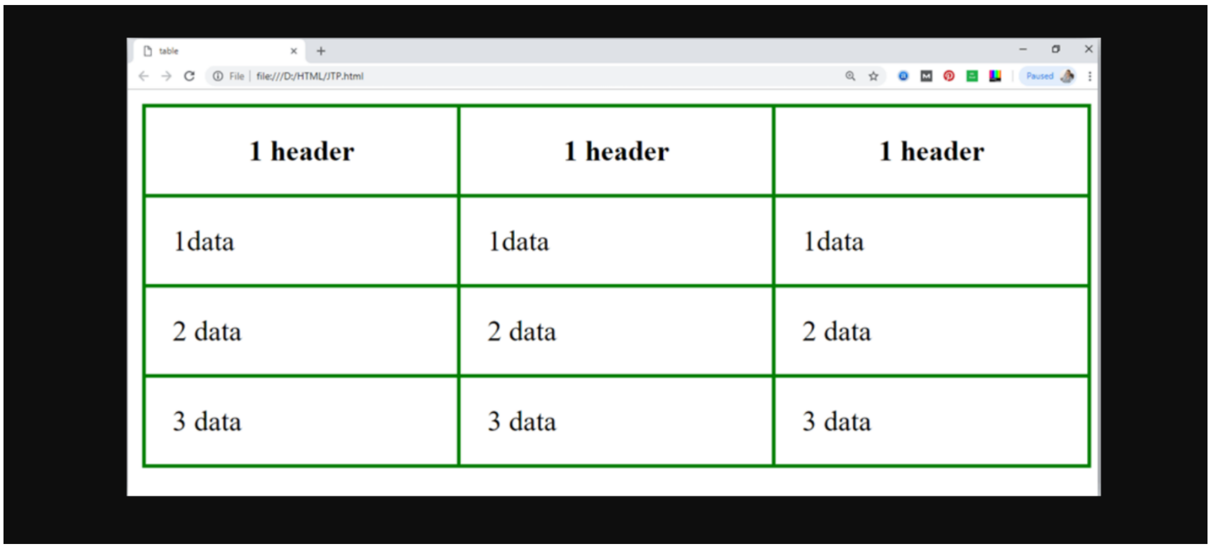
Each table row starts with a `<tr>` and ends with a `</tr>` tag.

The `<th>` tag defines a header cell in an HTML table.

The `<td>` tag defines a standard data cell in an HTML table.

In HTML tables are created in row by row.

Example:



1 header	1 header	1 header
1data	1data	1data
2 data	2 data	2 data
3 data	3 data	3 data

The `<thead>` tag is used to group header content in an HTML table.

The `<tbody>` tag is used to group the body content in an HTML table.

The `<tfoot>` tag is used to group footer content in an HTML table.

The `<caption>` tag defines a table caption.

The `<caption>` tag must be inserted immediately after the <table> tag.

To make a cell span over multiple columns, use the `colspan` attribute.

To make a cell span over multiple rows, use the `rowspan` attribute.

The `<colgroup>` tag specifies a group of one or more columns in a table for formatting.

The `<colgroup>` tag is useful for applying styles to entire columns, instead of repeating the styles for each cell, for each row.

The `<col>` tag specifies column properties for each column within a <colgroup> element.

The `<col>` tag is useful for applying styles to entire columns, instead of repeating the styles for each cell, for each row.

Attributes in col tag:

span	*number*	Specifies the number of columns a <col> element should span

HTML <a></a> (anchor) tag:

The `<a>` tag defines a hyperlink, which is used to link from one page to another.

The most important attribute of the `<a>` element is the `href` attribute, which indicates the link's destination.

By default, links will appear as follows in all browsers:

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

NOTE: If the `<a>` tag has no `href` attribute, it is only a placeholder for a hyperlink.

Attributes for a tag:

1)href

2)target

3)download

1)href attribute

The `href` attribute specifies the URL of the page the link goes to.

If the `href` attribute is not present, the `<a>` tag will not be a hyperlink.

Syntax

```
<a href="URL"></a>
```

Example:

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

2)target attribute

The target attribute specifies where to open the linked document.

Value	Description
_blank	Opens the linked document in a new window or tab
_self	Opens the linked document in the same frame as it was clicked (this is default)
_parent	Opens the linked document in the parent frame
_top	Opens the linked document in the full body of the window

3)download attribute

The download attribute specifies that the target (the file specified in the href attribute) will be downloaded when a user clicks on the hyperlink.

The optional value of the download attribute will be the new name of the file after it is downloaded. There are no restrictions on allowed values, and the browser will automatically detect the correct file extension and add it to the file (.img, .pdf, .txt, .html, etc.).

If the value is omitted, the original filename is used.

Examples:

```
<a href="/images/schoolimage.jpg" download>click</a>

<a href="/images/schoolimage.jpg" download="old_picture">click here</a>



<a href="#top"></a> it helps to move the top section of the web page.
```

HTML <img> Tag:

The `<img>` tag is used to embed an image in an HTML page.

Images are not technically inserted into a web page; images are linked to web pages. The `<img>` tag creates a holding space for the referenced image.

The `<img>` tag has two required attributes:

- src - Specifies the path to the image
- alt - Specifies an alternate text for the image, if the image for some reason cannot be displayed

Note: Also, always specify the width and height of an image. If width and height are not specified, the page might flicker while the image loads.

HTML <audio> Tag:

The `<audio>` tag is used to embed sound content in a document, such as music or other audio streams.

The text between the `<audio>` and `</audio>` tags will only be displayed in browsers that do not support the `<audio>` element.

There are three supported audio formats in HTML: MP3, WAV, and OGG.

Attribute	**Value**	**Description**
autoplay	autoplay	Specifies that the audio will start playing as soon as it is ready
controls	controls	Specifies that audio controls should be displayed (such as a play/pause button)
loop	loop	Specifies that the audio will start over again, every time it is finished

muted	muted	Specifies that the audio output should be muted

HTML <video> Tag:

Same as audio tag

The <figure> tag:

The <figure> tag specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.

The <figcaption> tag:

The <figcaption> tag defines a caption for a <figure> element.

The <figcaption> element can be placed as the first or last child of the <figure> element.

The <details> tag:

The <details> tag specifies additional details that the user can open and close on demand.

The <details> tag is often used to create an interactive widget that the user can open and close. By default, the widget is closed. When open, it expands, and displays the content within.

The <summary> tag:

The <summary> tag defines a visible heading for the <details> element. The heading can be clicked to view/hide the details.

Note: The <summary> element should be the first child element of the <details> element.

The <form> tag :

The <form> tag is used to create an HTML form for user input.

The <input> tag:

The <input> tag specifies an input field where the user can enter data.

The <input> element is the most important form element.

The <input> element can be displayed in several ways, depending on the type attribute.

The different input types are as follows:

```
<input type="text"> (default value)
<input type="email">
<input type="number">
<input type="password">
<input type="tel">
<input type="date">
<input type="checkbox">
<input type="radio">
<input type="range">
<input type="submit">
<input type="search">
<input type="reset">
<input type="url">
<input type="button">
```

HTML tags are classified into two types.

- **Semantic**
- **Non-Semantic**

Semantic Elements:

Semantic elements have meaningful names that tell about the type of content. For example header, footer, table, … etc. HTML5 introduces many semantic elements as mentioned below which make the code easier to write and understand for the developer as well as instruct the browser on how to treat them.

- <article>
  It contains independent content which doesn't require any other context blog Post, Newspaper Article, etc.
- <aside>
  It is used to place content in a sidebar i.e. aside from the existing content. It is related to surrounding content.
- <details>
  "details" defines additional details that the user can hide or view. "summary" defines a visible heading for a "details" element.
- <figcaption>
  These are used to add an image to a web page with a small description.
- <figure>
  These are used to add an image to a web page with a small description.
- <footer>
  Footer located at the bottom of any article or document, they can contain contact details, copyright information etc. There can be multiple footers on a page.
- <header>
  As the name suggests, it is for the header of a section introductory of a page. There can be multiple headers on a page.
- <main>
  It defines the main content of the document. The content inside the main tag should be unique.

- <nav>
  It is used to define a set of navigation links in the form of a navigation bar or nav menu.
- <section>

  A page can be split into sections like Introduction, Contact Information, Details, etc and each of these sections can be in a different section tag.

Non-Semantic Elements:

Tags like div, and span fall under the Non-Semantic categories as their names don't tell anything about what kind of content is present inside them.

CSS

CSS stands for Cascading Style Sheet.

Used to give styling to the web pages which is structured by the HTML language.

There are three ways in CSS to Style the Web pages

1. Inline CSS
2. Internal CSS
3. External CSS
4. Import.

1. **Inline CSS:** Inline CSS code is written in the opening tag of the HTML element by using style attribute.

Syntax:
```
<p style=" "> </p>
```

2. **Internal CSS:** Internal CSS code is written in the head section of an HTML element using <style> tag.

Syntax:
```
<head>
        <style>
            tag_name{
                    property: value;
                  }
        <style>
    </head>
```

3. **External CSS:** External CSS styling can be done by creating a external CSS file with file_name.css extention and providing the link between that CSS file to HTML file by <link> tag in the head section of HTML element.

Syntax:
```
<head>
        <link rel="stylesheet" href=" file_name.css">
```

</head>

4. @import: This Styling can be done by linking one CSS file to another CSS file.

Syntax: @import url(file_name.css)

Import should be written in very first line of the CSS file which has to be linked.

NOTE: The first priority is inline CSS

But the priorities of the Internal and External CSS varies accordingly depending on the declared position

```
                    if,     <link rel="stylesheet href="index.css">
              <style>  </style>
```

Here, the first priority is for Internal CSS.

```
if,     <style>  </style>
<link rel="stylesheet href="index.css">
```

Here, the first priority is for External CSS.

SELECTOR

Selectors are used to select the particular HTML element and to style them.

There are five types of Selectors

1. Simple Selectors
2. Combinator Selectors
3. Attribute Selectors
4. Pseudo Element Selectors.
5. Pseudo Classes Selectors.

Simple Selectors

Simple selectors style the HTML element in five ways:

1. Id Selector.
2. Class Selector.
3. Tagname.
4. Groupname.
5. Universe.

Id selectors:

Id selector targets only individual element in the HTML document.

Prefix symbol used for id selector is # (hash)

Example:

```
#demo{
color: red;
background: yellow;
}
```

Class Selector

Class selector targets multiple elements in the HTML document.

Prefix symbol used for id selector is . (dot)

Example:

```
.test{
color: red;
background: yellow;
}
```

Tagname Selector

Tagname selector targets the HTML elements by tag name.

There is no symbol used in tagname selectors.

Example:
```
        h1{
        color: magenta;
        background: green;
    }
h2{
        color: red;
        background: black;
    }
h3{
        color: yellow;
        background: orange;
    }
```

Grouping Selector

Groupname selector targets a group of HTML elements by tagname separated by commas.

Example:
```
        p, div, h4{
        color: magenta;
        background: green;
    }
```

Universal Selector

Universal Selectors targets every HMTL element.

The symbol of Universal Selector is *

There is no selector is declared in css.

Example:
```
        *{
        color: yellow;
        background: magenta;
    }
```

Color Property : Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

1. An RGB color value represents RED, GREEN, and BLUE light sources.

2. RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color. `rgba(`*`red, green, blue, alpha`*`)`. The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all)

3. HEX, A hexadecimal color is specified with: #RRGGBB, where the RR (red), GG (green) and BB (blue) hexadecimal integers specify the components of the color. Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255). Sometimes you will see a 3-digit hex code in the CSS source.The 3-digit hex code is a shorthand for some 6-digit hex codes.

4. HSL stands for hue, saturation, and lightness.**`hsl(`*`hue, saturation, lightness`*`).`** Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
5. HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity for a color. The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all). `hsla(`*`hue, saturation, lightness, alpha`*`)`

Note: CSS/HTML support 140 standard color names.

Example: color: color-name | rgb() | hex | hsl () | rgba() | hsla();

CSS background properties

The CSS background properties are used to add background effects for elements.

1. `background-color`

    - The `background-color` property specifies the background color of an element.
    - Example: `background-color: lightblue;`

2. background-image

- The background-image property specifies an image to use as the background of an element.
- By default, the image is repeated so it covers the entire element

3. background-repeat: repeat-x | repeat-y | no-repeat ;
4. background-position : right top | center | right bottom
5. The background-attachment property specifies whether the background image should scroll or be fixed (will not scroll with the rest of the page):

   - background-attachment: fixed | scroll;

6. background-size: cover | contain;
7. background, Use the shorthand property to set the background properties in one declaration.

   Example:

```
background: #ffffff url("img_tree.png") no-repeat right top;
```

BOX Model

Box Model is essentially a box that wraps around every HTML element. Box Model is used to design and layout.

The CSS Box Model consists of Margin, Border, Padding and the actual Content.

Content: It is the content of the HTML element.

Padding: It is the space after the content.

Border: It is the Border of the HTML element.

Margin: It is the space after the Border of an HTML element.

Padding and Margin allow up to four values.

Combinator Selector

Combinator Selectors are used to style the combination of HTML elements.

There are four Combinator Selectors:

1. Descendent Selectors
2. Child Selectors
3. Adjacent Sibling Selectors
4. General Sibling Selectors

1. Descendent Selectors: This Selector targets both direct and indirect children of a parent tag. The symbol is (space)

Syntax: parent_tag child_tag{property: value;}

2. Child Selectors: This Selector targets the only the direct children of the parent tag. The symbol is > (greater than)

Syntax: parent_tag > child_tag{property: value;}

3. Adjacent Selectors: This Selector targets the element which is the first sibling of the targeted HTML tag or element.

(or)

This Selector targets the element which is exactly adjacent to the targeted HTML tag or element.

The symbol is + (plus)

Syntax: parent_tag + target_tag{property: value;}

4. General Sibling Selectors: This Selector targets all the siblings of the target HTML tag or element.

(or)

This Selector targets all the adjacent tags of the target HTML tag or element. The symbol is ~ (tilde)

Syntax: parent_tag ~ target_tag{property: value;}

CSS Pseudo Classes Selectors.

A Pseudo class in CSS is used to define the special state of an element. It can be combined with a CSS selector to add an effect to existing elements based on their states. For Example, changing the style of an element when the user hovers over it, or when a link is visited. All of these can be done using Pseudo Classes in CSS.

Note that pseudo-class names are not case-sensitive.

Syntax:

```
selector: pseudo-class{
   property: value;
}
```

There are many Pseudo-classes in CSS but the ones that are most commonly used are as follows:

1):hover

2):active

3):focus

4):visited

5):first-child

6):nth-child(n)

1) :hover :This pseudo-class is used to add a special effect to an element when our mouse pointer is over it.

ex:

```
<style>
.box {
         background-color: yellow;
         width: 300px;
         height: 200px;
         margin: auto;
         font-size: 40px;
         text-align: center;
      }
.box:hover {
         background-color: orange;
      }

</style>
<body>
```

```
 <div class="box">
      My color changes if you hover over me!
   </div>
</body>
```

2):active Pseudo-class: This pseudo-class is used to select an element that is activated when the user clicks on it.

ex:

```
<style>
 .box{
      background-color: yellow;
      width: 300px;
      height: 200px;
      margin: auto;
      font-size: 40px;
      text-align: center;
   }
.box:active{
      background-color: orange;
   }

</style>
<body>
 <div class="box">
      My color changes if you hover over me!
   </div>
</body>
```

3):focus Pseudo-class: This pseudo-class is used to select an element that is currently focused by the user. It works on user input elements used in forms and is triggered as soon as the user clicks on it.

ex:

```
input:focus{
      background-color: grey;
    }
```

4):visited Pseudo-class: This pseudo-class is used to select the links which have been already visited by the user.

```
ex: a:visited{
      color: red;
    }
```

5) **:first-child** pseudo-class matches a specified element that is the first child of another element.

```
ex:p:first-child {
  color: blue;
}
```

6) **:nth-child(n)** selector matches every element that is the nth child of its parent.

n can be a number, a keyword (odd or even), or a formula (like an + b).

```
ex:tr:nth-child(odd) {
  background: red;
}

tr:nth-child(even) {
  background: lightgreen;
}
```

CSS Pseudo Element Selectors.

A CSS pseudo-element is a keyword added to a selector that lets you style a specific part of the selected elements. For Example, Styling the first letter or line of an element, and Inserting content before or after the content of an element. All of these can be done using Pseudo Elements in CSS.

Note that in contrast to pseudo-elements, pseudo-classes can be used to style an element based on its state.

Syntax:

```
selector::pseudo-element {
    property: value;
}
```

There are many Pseudo Elements in CSS but the ones which are most commonly used are as follows:

1)::first-line

2)::first-letter

3)::before

4)::after

5)::selection

1)::first-line Pseudo-element applies styles to the first line of a block-level element. Note that the length of the first line depends on many factors, including the width of the element, the width of the document, and the font size of the text. Note that only a few properties are applied for first-line pseudo-element like font properties, color properties, background properties, word-spacing, letter-spacing, text-decoration, vertical-align, text-transform, line-height, clear, etc.

ex:

```
<style>
      body {
         background-color: whitesmoke;
         color: green;
         font-size: large;
         text-align: center;
      }

      p::first-line {
         color: Red;
         font-weight: bold;
```

```
    }
  </style>
        <body>
  <p>
    This is a paragraph using first-line
    pseudo-element to style first line of
    the paragraph. Content in the first
    line turns red and becomes bold.
  </p>
</body>
```

2)::first-letter Pseudo-element applies styles to the first letter of the first line of a block-level element, but only when not preceded by other content (such as images or inline tables). Note that only a few properties are applied for first-line pseudo-element like font properties, color properties, background properties, word-spacing, letter-spacing, text-decoration, vertical-align, text-transform, line-height, clear, etc.

```
ex:  <style>
    body {
      background-color: whitesmoke;
      color: green;
      font-size: large;
      text-align: center;
    }

    p::first-letter {
      color: Red;
      font-size: 30px;
      font-weight: bold;
    }
  </style>
        <body>
  <p>
    This is a paragraph using first-letter
```

```
        pseudo-element to style first letter

        of the paragraph. first-letter element

        turned the first letter of this paragraph

        to red and made it bold.

    </p>

</body>
```

3)::before Pseudo-element creates a pseudo-element that is the first child of the selected element. It is often used to add cosmetic content to an element with the content property. It is inline by default.

ex:

```
p::before {

          content: '"';

          color: red;

          font-size: 30px;

      }
```

4)::after Pseudo-element creates a pseudo-element that is the last child of the selected element. It is often used to add cosmetic content to an element with the content property. It is inline by default.

ex:

```
 p::after {

          content: '"';

          color: red;

          font-size: 30px;

      }
```

5)::selection Pseudo-element applies styles to the part of a document that has been highlighted by the user such as clicking and dragging the mouse across the text.

ex:

```
p::selection {

          color: red;

          background-color: green;

          font-size: 30px;

      }
```

```
::selection {
   color: green;
   background-color: red;
   font-size: 30px;
}
```

CSS [attribute] Selector

The `[attribute]` selector is used to select elements with a specified attribute.

Example:

```
a[target] {
  background-color: yellow;
}
```

CSS [attribute="value"] Selector

The `[attribute="value"]` selector is used to select elements with a specified attribute and value.

Example:

```
a[target="_blank"] {
  background-color: yellow;
}
```

CSS [attribute~="value"] Selector

The `[attribute~="value"]` selector is used to select elements with an attribute value containing a specified word.

Example:

```
[title~="flower"] {
  border: 5px solid yellow;
}
```

CSS [attribute|="value"] Selector

The `[attribute|="value"]` selector is used to select elements with the specified attribute, whose value can be exactly the specified value, or the specified value followed by a hyphen (-).

Note: The value has to be a whole word, either alone, like class="top", or followed by a hyphen(-), like class="top-text".

Example:

```
[class|="top"] {
  background: yellow;
}
```

CSS [attribute^="value"] Selector

The `[attribute^="value"]` selector is used to select elements with the specified attribute, whose value starts with the specified value.

Note: The value does not have to be a whole word!

Example:

```
[class^="top"] {
  background: yellow;
}
```

CSS [attribute$="value"] Selector

The `[attribute$="value"]` selector is used to select elements whose attribute value ends with a specified value.

Note: The value does not have to be a whole word!

Example:

```
[class$="test"] {
  background: yellow;
}
```

CSS [attribute*="value"] Selector

The `[attribute*="value"]` selector is used to select elements whose attribute value contains a specified value.

Note: The value does not have to be a whole word!

Example:

```
[class*="te"] {
  background: yellow;
}
```

Note: The attribute selectors can be useful for styling forms without class or ID.

Example:

```
input[type="text"] {
  width: 150px;
  background-color: yellow;
}

input[type="button"] {
  width: 120px;
background-color: lightblue;
}
```

The display Property

The `display` property is used to specify how an element is shown on a web page.

Every HTML element has a default display value, depending on what type of element it is. The default display value for most elements is `block` or `inline`.

The `display` property is used to change the default display behavior of HTML elements.

Block-level Elements:

A block-level element ALWAYS starts on a new line and takes up the full width available .

Examples of block-level elements:

- <div>
- <h1> to <h6>
- <p>
- <form>
- <header>
- <footer>
- <section>
- <li>

- <figure>
- <pre>
- <nav>
- <hr>
- <ul>
- <ol>
- <table>

Inline Elements:

An inline element DOES NOT start on a new line and only takes up as much width as necessary.

Examples of inline elements:

- <span>
- <a>
- <button>
- <select>
- <textarea>
- All formatting tags

The display Property Values

Value	Description
1. Inline	1. Displays an element as an inline element
2. Block	2. Displays an element as a block element
3. inline-block	3. Displays an element as an inline-level block container. The element itself is formatted as an inline element, but you can apply height and width values
4. none	4. The element is completely removed.
5. flex	5. Displays an element as a block-level flex container
6. grid	6. Displays an element as a block-level grid container

Inline Block Elements

The display value of inline-block works similarly to inline with one exception: the height and width of that element become modifiable.

`visibility:hidden;` also hides an element.

However, the element will still take up the same space as before. The element will be hidden, but still affect the layout:

Difference between display:none and visiblity: hidden

visibility:hidden hides the element, but it still takes up space in the layout.

display:none removes the element from the document. It does not take up any space.

display:flex;

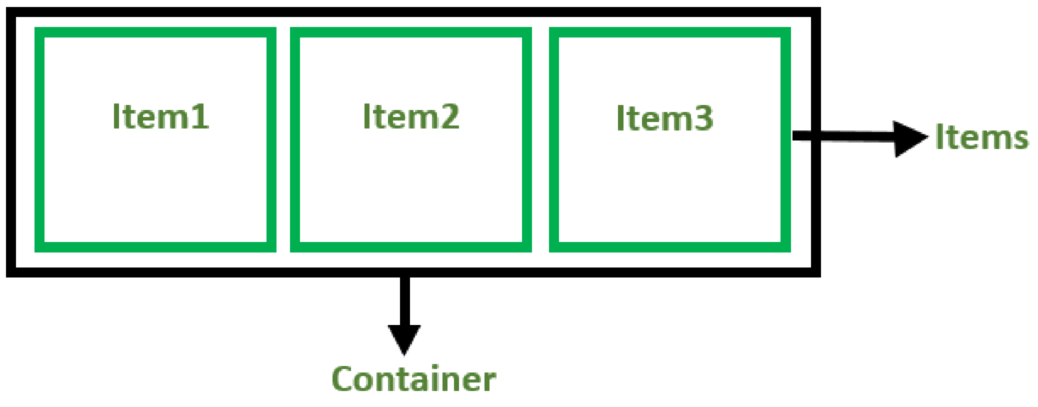
The **flexbox** or flexible box model in CSS is a one-dimensional layout model that has flexible and efficient layouts with distributed spaces among items to control their alignment structure ie., it is a layout model that provides an easy and clean way to arrange items within a container. Flexbox can be useful for creating small-scale layouts & is responsive and mobile-friendly

Features of flexbox:

- A lot of flexibility is given.
- Arrangement & alignment of items.
- Proper spacing
- Order & Sequencing of items.
- Bootstrap 4 is built on top of the flex layout.

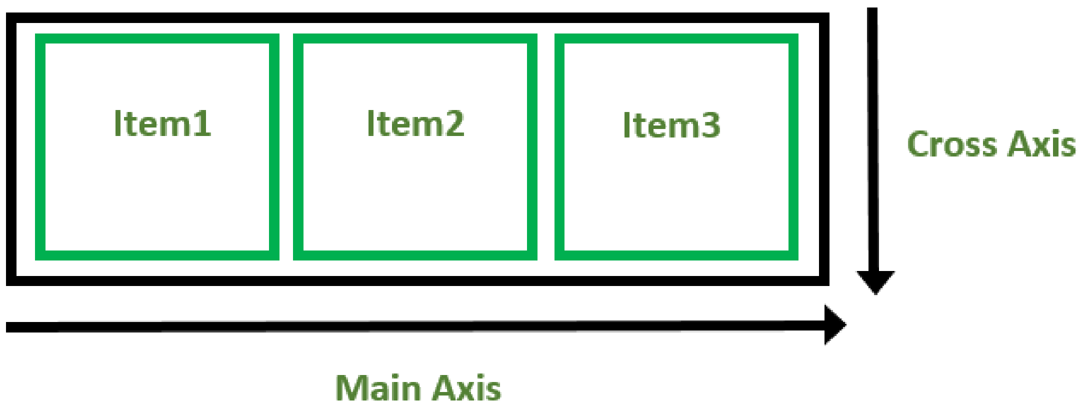
There are 2 main components of the Flexbox:

- **Flex Container**: The parent “div” which contains various divisions is called a flex container.
- **Flex Items**: The items inside the container “div” are flex items.



Flexbox Axes: While working with Flexbox, we deal with 2 axes:

- Main Axis
- Cross Axis



Properties of flexbox:
Parent properties:

- **display** defines a flex container – flex formatting context.
- **flex-direction** defines the main axis inside the container.
- **flex-wrap** allows flex items to wrap onto more than one line.
- **justify-content** allows items to align along main axis.
- **align-items** aligns multiple lines of items on the cross axis.

Children/Flex-items properties:

- **flex-grow** allows an item to fill up the available free space.
- **flex-shrink** allows an item to shrink if there is no enough free space available.
- **flex-basis** defines the size of an item before space is distributed.

CSS Positions

The **position** property in CSS tells about the method of positioning for an element or an HTML entity and the positioning of an element can be done using the top, right, bottom, and left properties.

There are five different position values:

- `static`
- `relative`
- `fixed`
- `absolute`
- `sticky`

Note: Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the `position` property is set first. They also work differently depending on the position value.

position: static;

HTML elements are positioned static by default.

Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with `position: static;` is not positioned in any special way; it is always positioned according to the normal flow of the page:

position: fixed;

An element with `position: fixed;` is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

position: sticky;

An element with `position: sticky;` is positioned based on the user's scroll position.

A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

position: relative;

An element with `position: relative;` is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

position: absolute;

An element with `position: absolute;` is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.

CSS z-index **Property**

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

An element with greater stack order is always in front of an element with a lower stack order.

Note: `z-index` only works on positioned elements (position: absolute, position: relative, position: fixed, or position: sticky) and flex items (elements that are direct children of display:flex elements).

Note: If two positioned elements overlap without a `z-index` specified, the element positioned last in the HTML code will be shown on top.

CSS Overflow

The `overflow` property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

The `overflow` property has the following values:

- `visible` - Default. The overflow is not clipped. The content renders outside the element's box
- `hidden` - The overflow is clipped, and the rest of the content will be invisible
- `scroll` - The overflow is clipped, and a scrollbar is added to see the rest of the content
- `auto` - Similar to `scroll`, but it adds scrollbars only when necessary

Note: The `overflow` property only works for block elements with a specified height.

overflow-x and overflow-y

The `overflow-x` and `overflow-y` properties specifies whether to change the overflow of content just horizontally or vertically (or both):

`overflow-x` specifies what to do with the left/right edges of the content.
`overflow-y` specifies what to do with the top/bottom edges of the content.

JAVASCRIPT

What is JavaScript?

Javascript is a high-level , object based ,programming language. Javascript is a interpreted, user friendly ,client-side scripting language.

History of javascript:

javascript developed by Brenden Eich-1995

First name of javascript --> MOCHA

next --> LIVESCRIPT

next--> JAVASCRIPT

FEATURES OF JAVASCRIPT:

1)Light Weight

2)intrepreted language

3)open source

4)cross platform

5)Dynamically typed language

6)Weakley typed language

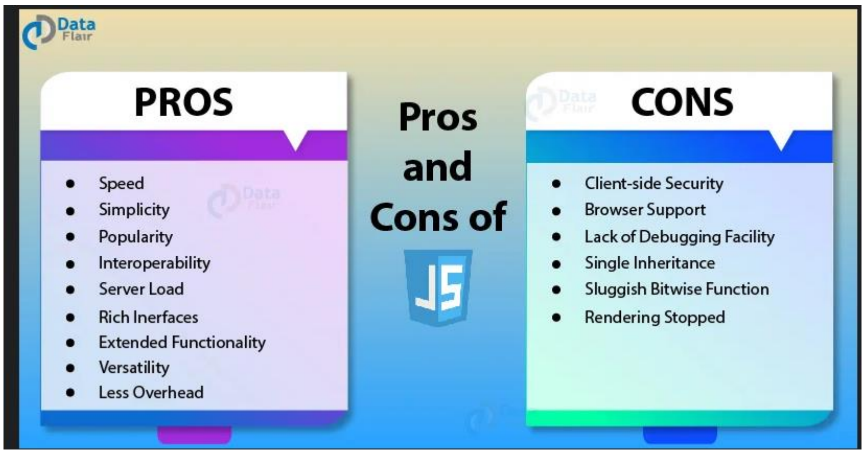
7)client-side scripting language

8)Synchronous language

Javascript characteristics:

1. *Client-side-Scripting language* - no compiler and without help of server logic we can update the data
2. *High-level language* - user-friendly
3. *Interpreted language* – line by line execution
4. *Single threaded* – one call stack and one heap area
5. *Loosely typed* – no strong syntax
6. *Dynamic language* – can change the datatype during the runtime
7. *Object-based language* – In JavaScript everything was object

Advantages and disadvantages of javaScript:



PROS

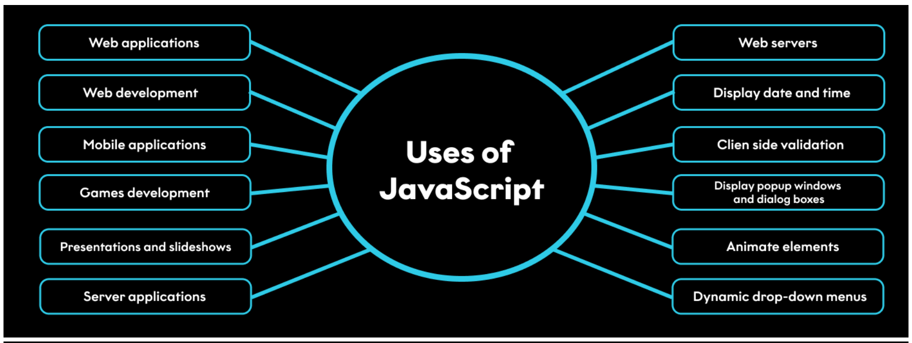
- Speed
- Simplicity
- Popularity
- Interoperability
- Server Load
- Rich Inerfaces
- Extended Functionality
- Versatility
- Less Overhead

Pros and Cons of JS

CONS

- Client-side Security
- Browser Support
- Lack of Debugging Facility
- Single Inheritance
- Sluggish Bitwise Function
- Rendering Stopped

Uses of JavaScript:



Difference between Java and JavaScript

Java	javascript
Java is a strongly typed language and variables must be declared first to use in the program. In Java, the type of a variable is checked at compile-time.	JavaScript is a loosely typed language and has a more relaxed syntax and rules.
Java is an object-oriented programming language primarily used for developing complex logic applications.	JavaScript is a scripting language used for creating interactive and dynamic web pages.
Java applications can run in any virtual machine(JVM) or browser.	JavaScript code used to run only in the browser, but now it can run on the server via Node.js.
Objects of Java are class-based even we can't make any program in java without creating a class.	JavaScript Objects are prototype-based.
Java program has the file extension ".Java" and translates source code into bytecodes which are executed by JVM(Java Virtual Machine).	JavaScript file has the file extension ".js" and it is interpreted but not compiled, every browser has the Javascript interpreter to execute JS code.
Java supports multithreading, which allows multiple threads of execution to run concurrently within a single program.	JavaScript does not support multithreading, although it can simulate it through the use of web workers.
Java has a rich set of libraries and frameworks for building enterprise applications, such as Spring, Hibernate	JavaScript has a wide variety of libraries and frameworks for building web applications, such as React, Angular, and Vue.

Java is mainly used for backend. Java is statically typed, which means that data types are determined at compile time. Java uses more memory	Javascript is used for the frontend and backend both. JavaScript is dynamically typed, which means that data types are determined at runtime. Javascript uses less memory.

Note: The file extension for the javascript is **.js**

Attaching the javascript file to html file:

We can attach the js file with html file in 2 ways.

1.**Internal Way**

2.External Way

1.Internal Way

By writing the code of javascript inside <script></script> tag

<u>**Syntax:**</u>

```
<body>
        <h1>hello world</h1>
        <script>
                // javascript code……….
        </script>
 </body>
```

2. External Way

1. To write javascript externally to html file, we need to create an external javascript file with extension as **.js (**ex: index.js**)**
2. After that link that javascript file to the html by using script tag.

   Example:

```
<body>
        <script src = "./index.js "></script>
</body>
```

Printing Statements in javascript:

In javascript we have 2 types of printing statements

1.document.write();

2.console.log();

1.document.write()

document.write() is a printing statement which is used to see the output in the web browser (client purpose)

2.console.log()

Console.log() is a printing statement which is used to see the output in the console window (developer view)

Here, console. is an object , dot is a period (or) access operator and log is a function and member of console. that accepts argument as data to print in console.

TOKENS:

In JavaScript, a token is the smallest unit of a program that is meaningful to the interpreter. JavaScript code is made up of various types of tokens, including

1. **Keywords:**
   Reserved words that have special meanings in the language. Examples include `var`, `if`, `else`, `for`, `while`, etc.
   NOTE: Every keyword must be in the lower case and it is not used as Identifiers.

2. Identifiers:

These are names that are used to identify variables, functions, and other objects. For example, the identifier `myVariable` could be used to identify a variable that stores the value 5.

Rulers: -an identifiers can't start with number. -We can't use keywords as identifiers. -Except $, _ no other Special Characteristics are allowed.

3.Literals:

Data which is used in the js programming is called as literals. For example, the literal `5` is a numeric literal, and the literal `"Hello, world!"` is a string literal.

4. Operators:

These are symbols that are used to perform operations on operands. For example, the operator `+` is used to add two numbers together.

JavaScript Variable

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

1. var x = 10;
2. var _value="sonoo";

Incorrect JavaScript variables

1. var 123=30;
2. var *aa=320;

Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value

Null	represents null i.e. no value at all
Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript non-primitive data types

The non-primitive data types are as follows:

Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JavaScript Operators

Operators are special symbols in JavaScript that are used to perform operations on values. There are many different types of operators, including arithmetic operators, comparison operators, logical operators, and assignment operators.

JavaScript Arithmetic Operators

Arithmetic Operators are used to perform arithmetic on numbers:

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

JavaScript Assignment

The **Assignment Operator** (`=`) assigns a value to a variable:

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example

==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10===20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!=="20" = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

Javascript Conditional Statements:

JavaScript If statement

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

1. if(expression){

```
//content to be evaluated
}
```

Example:

```
<script>
var a=20;
if(a>10){
document.write("value of a is greater than 10");
}
</script>
```

JavaScript If...else Statement

It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

```
if(expression){
//content to be evaluated if condition is true
}
else{
//content to be evaluated if condition is false
}
```

Example:

```
<script>
var a=20;
if(a%2==0){
document.write("a is even number");
}
else{
document.write("a is odd number");
}
</script>
```

JavaScript If...else if statement

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

```
if(expression1){
//content to be evaluated if expression1 is true
}
else if(expression2){
//content to be evaluated if expression2 is true
}
else if(expression3){
//content to be evaluated if expression3 is true
}
else{
//content to be evaluated if no expression is true
}
```

Example:

```
<script>
var a=20;
if(a==10){
document.write("a is equal to 10");
}
else if(a==15){
document.write("a is equal to 15");
}
else if(a==20){
document.write("a is equal to 20");
}
else{
document.write("a is not equal to 10, 15 or 20");
}
</script>
```

Switch Statement:

JavaScript Switch statement

Switch Statement Execute the block of code depend on the different cases..

Syntax:

```
 switch(expression) {
case n:
 // code block
  break;
case n:
  //code block
  break;
default:
  default: //code block
}
```

Note:

if we not give the break keyword then ,even the condition is satisfied it will execute the remaining blocks.

Break Keyword: It Will Break the Execution At that particular point ..

Example:

```
const food = "nuts";

switch (food) {
  case "cake":
    console.log("I like cake");
    break;
  case "pizza":
    console.log("I like pizza");
    break;
  default:
    console.log("I like all foods");
    break;
  case "ice cream":
    console.log("I like ice cream");
    break;
}
```

Javascript var,let,const

	Declaration	Initialization	Declaration and initialization	Re-Declaration	Re-Initialization	Re-declaration & Re-Initialization
var	✓	✓	✓	✓	✓	✓
let	✓	✓	✓	✗	✓	✗
const	✗	✗	✓	✗	✗	✗

JavaScript Loops:

JavaScript Loops are powerful tools for performing repetitive tasks efficiently. Loops in JavaScript execute a block of code again and again while the condition is true.

- for Loop
- while Loop
- do-while Loop
- for-in Loop
- for-of Loop

1)JavaScript for Loop

The JS for loop provides a concise way of writing the loop structure. The for loop contains initialization, condition, and increment/decrement in one line thereby providing a shorter, easy-to-debug structure of looping.

Syntax

```
for (initialization; testing condition; increment/decrement) {
    statement(s)
}
```

- **Initialization condition:** It initializes the variable and mark the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
- **Test Condition:** It is used for testing the exit condition of a for loop. It must return a boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
- **Statement execution:** Once the condition is evaluated to be true, the statements in the loop body are executed.
- **Increment/ Decrement:** It is used for updating the variable for the next iteration.
- **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

Example:

```
for (x = 2; x <= 4; x++) {
    console.log("Value of x: " + x);
}
```

2)JavaScript while Loop

The JS while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax

```
while (boolean condition) {
    loop statements...
}
```

- While loop starts with checking the condition. If it is evaluated to be true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason, it is also called the **Entry control loop**
- Once the condition is evaluated to be true, the statements in the loop body are executed. Normally the statements contain an updated value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

Example:

```
let val = 1;
```

```
while (val < 6) {
    console.log(val);
    val += 1;
}
```

3)JavaScript do-while Loop

The JS do-while loop is similar to the while loop with the only difference is that it checks for the condition after executing the statements, and therefore is an example of an **Exit Control Loop.** It executes loop content at least once event the condition is false.

Syntax

```
do {
    Statements...
}
while (condition);
```

- The do-while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements and update of the variable value, the condition is checked for a true or false value. If it is evaluated to be true, the next iteration of the loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked and therefore is an example of the exit control loop.

Example:

```
let test = 1;


do {

        console.log(test);

        test++;

} while(test <= 5)
```

4) JavaScript for...in loop

There are also other types of loops. The `for..in` loop in JavaScript allows you to iterate over all property keys of an object.

The syntax of the `for...in` loop is:

```
for (key in object) {
    // body of for...in
}
```

Note: In each iteration of the loop, a key is assigned to the `key` variable. The loop continues for all object properties.

Example:

```
const student = {
    name: 'Monica',
    class: 7,
    age: 12
}
// using for...in
for ( let key in student ) {
    // display the properties
    console.log(`${key} => ${student[key]}`);
}
```

for...in with Strings

You can also use `for...in` loop to iterate over string values.

For example,

```
const string = 'code';

// using for...in loop
for (let i in string) {
    console.log(string[i]);
}
```

for...in with Arrays

You can also use `for...in` with arrays. For example

```
// define array
const arr = [ 'hello', 1, 'JavaScript' ];

// using for...in loop
for (let x in arr) {
    console.log(arr[x]);
}
```

Note: You should not use `for...in` to iterate over an array where the index order is important.

One of the better ways to iterate over an array is using the `for...of` loop.

JavaScript for...of loop

The `for...of` loop was introduced in the later versions of **JavaScript ES6**.

The `for..of` loop in JavaScript allows you to iterate over iterable objects (arrays, sets, maps, strings etc).

The syntax of the `for...of` loop is:

```
for (element of iterable) {
   // body of for...of
}
```

- **iterable** - an iterable object (array, set, strings, etc).
- **element** - items in the iterable

for...of with Arrays

The `for..of` loop can be used to iterate over an array. For example,

```
// array
const students = ['John', 'Sara', 'Jack'];

// using for...of
for ( let element of students ) {

    // display the values
    console.log(element);
}
```

for...of with Strings

You can use for...of loop to iterate over string values. For example,

```
// string
const string = 'code';

// using for...of loop
for (let i of string) {
    console.log(i);
}
```

for...of Vs for...in

for...of	for...in
The for...of loop is used to iterate through the values of an iterable.	The for...in loop is used to iterate through the keys of an object.
The for...of loop cannot be used to iterate over an object.	You can use for...in to iterate over an iterable such arrays and strings but you should avoid using for...in for iterables.

JavaScript Functions:

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

A JavaScript function is executed when "something" invokes it (calls it).

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: **{}**

Example:

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return:

When JavaScript reaches a `return` statement, the function will stop executing.If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Why Functions?

With functions you can reuse code

You can write code that can be used many times.

You can use the same code with different arguments, to produce different results.

Types of Functions in JS:

1. Anonymous function
2. Named function
3. Function with expression
4. Nested function
5. Immediate invoking function
6. Arrow function
7. Higher order function

8. Callback function

1.Anonymous function

A function without name is known as Anonymous function

Syntax :

```
function(parameters) {
                 // function body
                }
```

2.Named Function

A function with name is called as named function

Syntax :

```
function functionName(parameters) {
          // function body
          }
```

3 . Function with expression

It is the way to execute the anonymous function

Passing whole anonymous function as a value to a variable is known as function with expression.

The function which is passed as a value is known as first class function

```
EX: let x=function(){
//block of code
}
x();
```

4.Nested function

A function which is declared inside another function is known as nested function.

Nested functions are unidirectional i.e., We can access parent function properties in child function but vice-versa is not possible.

The ability of js engine to search a variable in the outer scope when it is not available in the local scope is known as lexical scoping or scope chaining.

Whenever the child function needs a property from parent function the closure will be formed and it consists of only required data.

A closure is a feature of JavaScript that allows inner functions to access the outer scope of a function. Closure helps in binding a function to its outer boundary and is created automatically whenever a function is created. A block is also treated as a scope since ES6. Since JavaScript is event-driven so closures are useful as it helps to maintain the state between events.

```
Function parent(){
        let a=10;
        function child(){
        let b=20;
        console.log(a+b);
}
child ();
}
parent ();
```

JavaScript currying

Calling a child function along with parent by using one more parenthesis is known as java script currying

Example:

```
Function parent () {
        let a=10;
        function child () {
        let b=20;
        console.log(a+b);
}
return child;
}
parent () ();   ==➔JavaScript currying
```

Immediate invoking function(IIF):

A function which is called immediately as soon as the function declaration is known as IIF

We can invoke anonymous function by using IIF

Example:

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

Arrow function:

It was introduced in ES6 version of JS.

The main purpose of using arrow function is to reduce the syntax.

If function has only one statement, then block is optional.
It is mandatory to use block if function consists of multiple lines of code or if we use return keyword in function.
Example:

```
Let x= (a, b) =>console.log(a+b);
Let y=(a,b)=>{return a + b };
```

Higher order function(HOF):

A function which accepts a function as a parameter is known as HOF
It is used to perform multiple operations with different values.

Example:

```
Function hof (a, b, task){
Let res=task(a,b);
return res;
};
Let add=hof(10,20,function(x , y){
return x + y;
}
Let mul = hof(10,20,function(x , y){
return x*y;
}
Console.log(add());
Console.log(mul());
```

Callback function:

A function which is passed as an argument to another function is known as callback function.
The function is invoked in the outer function to complete an action
Example :

```
function first(){
Console.log("first");
}
function second(){
Console.log("third");
}
function third(callback){
Console.log("second");
Callback()
}
first();
third(second);
```

STRING CONCEPT IN JAVASCRIPT

STRING:-Collection of characters (or) bunch of characters we called it as string

String methods:-
String.length
String.slice()
String.substring()
String.substr()
String.replace()
String.replaceAll()
String.toUpperCase()
Strin. toLowerCase
Strin.concat()
String.trim()
String.trimStart()
trimEnd()
padStart()
padEnd()
String charAt()
String charCodeAt()
String split()

JAVASCRIPT ARRAYS

ARRAYS:- Array is collection of different elements.Array is heterogrnous in nature.

- In javascript we can create array in 3 ways.
  1. By array literal

2.By using an Array constructor (using new keyword)

1) JavaScript array literal:-

- The syntax of creating array using array literal is: var arrayname=[value1,value2.....valueN];
- As you can see, values are contained inside [ ] and separated by , (comma).
- The .length property returns the length of an array.

Ex:-**<script>**

var emp=["Sonoo","Vimal","Ratan"]; for (i=0;i<**emp.length**;i++){ document.write(emp[i] + "**<br/>**");
}
</script>

2) JavaScript array constructor (new keyword):-
Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.
Ex:-
<script>
var emp=new Array("Jai","Vijay","Smith"); for (i=0;i<**emp.length**;i++){ document.write(emp[i] + "**<br>**");
}
</script>
Output:- Jai
Vijay Smith

Java script Array methods:

push() : It will insert an element of an array at the end
unshift() : It will insert an element of an array at the first
pop() : It will remove elements of array from the end
shift() : It will remove elements of array from the start
indexOf() :It will return a index of particular element
Includes() : It will check whether the particular element is present in array or not.
At() : It will return the element which is present in particular index.
Slice() : It will give slice of an array and it will not affect the original array.
Splice() : It is used to add or remove elements from an array
Join() : It is used to join all elements of an array into a string.
Concat() : It is used to join/concat two or more arrays.
toString() : It converts all the elements in an array into a single string and returns that string.
Split() : It is used to split a string into an array.
Reverse() : It is used to reverse an array.
Array.from() : It will convert string into an array

JAVASCRIPT OBJECTS

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.

The syntax to declare an object is:

```
const object_name = {
   key1: value1,
  key2: value2
}
```

Here, an object `object_name` is defined. Each member of an object is a **key: value** pair separated by commas and enclosed in curly braces `{}`.

Example:

```
// object creation
const person = {
    name: 'John',
age: 20
};
console.log(typeof person); // object
```

Accessing Object Properties

You can access the **value** of a property by using its **key**.

1. Using dot Notation

Here's the syntax of the dot notation.

```
objectname.key
```

Example:

```
const person = {
    name: 'John',
    age: 20,
};


// accessing property
console.log(person.name); // John
```

2. Using bracket Notation

Here is the syntax of the bracket notation.

objectName[“key”]

Example:

```
const person = {
    name: 'John',
    age: 20,
};
// accessing property
console.log(person["name"]); // John
```

JavaScript Nested Objects

An object can also contain another object. For example,

```
// nested object
const student = {
    name: 'John',
    age: 20,
    marks: {
        science: 70,
        math: 75
    }
}
// accessing property of student object
console.log(student.marks); // {science: 70, math: 75}
// accessing property of marks object
console.log(student.marks.science); // 70
```

JavaScript Object Methods

In JavaScript, an object can also contain a function. For example,

```
const person = {

    name: 'Sam',

    age: 30,

    // using function as a value

    greet: function() { console.log('hello') }

}

person.greet(); // hello
```

How to Modify Properties in an Object

You can also use dot notation to modify properties in an object. The below code shows that the property `name` with the pair value `sam` will be changed to `samantha` when modified using dot notation.

```
const person = {

    name: 'Sam',

    age: 30,

    // using function as a value

    greet: function() { console.log('hello') }

}
```

```
person.name = "Samantha";
console.log(person);
```

How to Delete Properties in an Object

It is very simple to delete a property in an object. JavaScript has a special keyword called "**delete**" that allows you to discard any properties you wish.

```
const person = {
    name: 'Sam',

    age: 30,

    // using function as a value

    greet: function() { console.log('hello') }

}

delete person.age;

console.log(person);
```

JavaScript Object Properties:

Object.entries()

The **Object.entries()** static method returns an array of a given object's own enumerable string-keyed property key-value pairs.

Example:

```
const obj = { foo: "bar", baz: 42 };
console.log(Object.entries(obj));   // [ ['foo', 'bar'], ['baz', 42] ]
```

Object.freeze()

The **Object.freeze()** static method *freezes* an object. Freezing an object prevents extensions and makes existing properties non-writable and non-configurable. A frozen object can no longer be changed: new properties cannot be added, existing properties cannot be removed

Example:

```
const obj = {

  prop: 42,

};

Object.freeze(obj);

obj.prop = 33;

// Throws an error in strict mode

console.log(obj.prop);
```

```
// Expected output: 42
```

Object.fromEntries()

The **Object.fromEntries()** static method transforms a list of key-value pairs into an object.

Example:

```
const arr = [
  ["0", "a"],
  ["1", "b"],
  ["2", "c"],
];
const obj = Object.fromEntries(arr);
console.log(obj); // { 0: "a", 1: "b", 2: "c" }
```

Object.keys()

The **Object.keys()** static method returns an array of a given object's own enumerable string-keyed property names.

Example:

```
const object1 = {
  a: 'somestring',
  b: 42,
  c: false,
};

console.log(Object.keys(object1));
// Expected output: Array ["a", "b", "c"]
```

Object.values()

The **Object.values()** static method returns an array of a given object's own enumerable string-keyed property values.

Example:

```
const object1 = {

  a: 'somestring',

  b: 42,

  c: false,

};
```

```
console.log(Object.values(object1));

// Expected output: Array ["somestring", 42, false]
```

Spread operator:

The spread operator helps us expand an iterable such as an array where multiple arguments are needed, it also helps to expand the object expressions. In cases where we require all the elements of an iterable or object to help us achieve a task, we use a spread operator.
Note: There can be more than one spread operator in javascript functions.
Syntax: `var var_name = [...iterable];`

Example 1:

```
var array1 = [10, 20, 30, 40, 50];

    var array2 = [60, 70, 80, 90, 100];
    var array3 = [...array1, ...array2];
   console.log(array3);
```

Output:

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Example 2:

```
    const obj = {
        firstname: "Divit",
        lastname: "Patidar",
    };
    const obj2 = { ...obj };
    console.log(obj2);
```

Output:

```
{

    firstname: "Divit",

    lastname: "Patidar"

}
```

Note:

There are three distinct places that accept the spread syntax:

- Function arguments list (myFunction(a, ...iterableObj, b))
- Array literals ([1, ...iterableObj, '4', 'five', 6])
- Object literals ({ ...obj, key: 'value' })

Rest parameter

The rest parameter is converse to the spread operator. while the spread operator expands elements of an iterable, the rest operator compresses them. It collects several elements. In functions when we require to pass arguments but were not sure how many we have to pass, the rest parameter makes it easier.
Note: There must be only one rest operator in javascript functions.
Note: The rest parameter has to be the last argument, as its job is to collect all the remaining arguments into an array. So having a function definition like the code below doesn't make any sense and will throw an error.
Syntax:

```
function function_name(...arguments) {

    statements;

}
```

Example:

```
 function average(...args) {
        console.log(args);
        var avg =
            args.reduce(function (a, b) {
                return a + b;
           }, 0) / args.length;
        return avg;
    }
    console.log("average of numbers is : "
        + average(1, 2, 3, 4, 5));
    console.log("average of numbers is : "
        + average(1, 2, 3));
```

Here is a table that summarizes the key differences between the two:

Feature	Rest Parameter	Spread Operator
Syntax	...	...
Usage	Used in function parameters	Used in function calls, array literals, and object literals
Purpose	Collects multiple elements into an array	Expands an array into its individual elements

Introduction to the DOM

The **Document Object Model (DOM)** is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document but the Document Object Model (DOM) representation allows it to be manipulated. As an object-oriented representation of the web page, it can be modified with a scripting language such as JavaScript.

Or

In simple words, it can be categorized as a **programming interface** for **HTML** as well as **XML** (e**X**tensible **M**arkup **L**anguage) documents. It defines the **logical structure** of the documents in the form of a tree of objects where each HTML element is represented as a node and it also describes the way these documents can be manipulated.

Now coming to the HTML DOM methods, there are **five** different methods in which users can access or manipulate HTML elements using JavaScript:

- HTML DOM getElementById() Method
- HTML DOM getElementsByClassName() Method
- HTML DOM getElementsByTagName() Method
- HTML DOM querySelector() Method
- HTML DOM querySelectorAll() Method

1) HTML DOM getElementById(): This method is used when developers have defined certain HTML elements with **IDs** that uniquely identify the same elements in the whole document. It returns an **Element object** which matches the specified ID in the method. If the ID does not exist in the document, it simply returns **null**.

Syntax:

```
document.getElementById(id);
```

Parameter: It has a single parameter as mentioned above and described below:

- **id:** The ID of the element to locate in the HTML document. It should be a case-sensitive string.

Return value: It returns the object corresponding to the passed ID in the method, or **null** if no match is found.

2) **getElementsByClassName():** This method is used when there are multiple HTML elements with the same class name. It returns a **collection** of all objects which match the specified class in the method.

Syntax:

```
document.getElementsByClassName(className);
```

Parameter: It has a single parameter as mentioned above and described below:

- **className**: The class name of the element(s) to locate in the HTML document. It should be a case-sensitive string.

Return value: It returns a collection of objects corresponding to the passed class name in the method.

3) **getElementsByTagName()**: The getElementsByTagName() returns a **HTMLCollection** of objects which match a **tag name** in the HTML document.

Syntax:

```
document.getElementsByTagName(tagName);
```

Parameter: It has a single parameter as mentioned above and described below:

- **tagName:** The tag name of the element(s) to locate in the HTML document. It should be a case-sensitive string.

Return value: It returns an **HTMLCollection** of objects corresponding to the passed tag name in the method.

4) **querySelector():** This method returns the **first match** of an element that is found within the HTML document with the specific selector. If there is no match**, null** is returned.

Syntax:

```
document.querySelector(selector);
```

Parameter: It has a single parameter as mentioned above and described below:

- **selector:** A string containing one or more selectors to match elements located in the HTML document.

Return value: It returns the first occurrence of the object corresponding to the passed selector in the method.

5) **querySelectorAll():** This method returns a static **NodeList** of all elements that are found within the HTML document with the specific selector.

Syntax:

```
document.querySelectorAll(selector);
```

Parameter: It has a single parameter as mentioned above and described below:

- **selector:** A string containing one or more selectors to match elements located in the HTML document.

Return value: It returns a **NodeList** of the objects corresponding to the passed selector in the method.

DOM createElement() Method:

In an HTML document, the **document.createElement()** is a method used to create the HTML element. The element specified using elementName is created or an unknown HTML element is created if the specified elementName is not recognized.

Syntax

```
let element = document.createElement("elementName");
```

In the above syntax, elementName is passed as a parameter in string form. elementName specifies the type of the created element. The nodeName of the created element is initialized to the elementName value. The document.createElement() returns the newly created element.

`innerText()`

Returns the text content of an element and all its child elements, excluding HTML tags and CSS hidden text. `innerText` also ignores HTML tags and treats them as part of the text.

`innerHTML()`

Returns the text content of an element, including all spacing and inner HTML tags. `innerHTML` recognizes HTML tags and renders the content according to the tags. `innerHTML` allows you to see exactly what's in the HTML markup contained within a string, including elements like spacing, line breaks, and formatting.

appendchild()

The **appendChild()** method enables us to **add** a node to the end of a specified **parent node's** child node list.

Syntax:

The syntax of the **appendchild()** method is:

1. parentNode.appendChild(childNode)

append()

The **Element.append()** method inserts a set of Node objects or string objects after the last child of the Element.

Syntax:

The syntax of the **append()** method is:

1. parentNode.append(childNode)

Differences from Node.appendChild():

- Element.append() can append several nodes and strings, whereas Node.appendChild() can only append one node.

```
append(param1)
append(param1, param2)
append(param1, param2, /* …, */ paramN)
```

Using removeChild() method
Using remove() method

Using the remove() method

The **`remove()`** method of JavaScript will remove the element from the document. The syntax for the remove method is shown below.

```
Obj.remove("element-name");
```

Using removeChild() method

The **`removeChild()`** method of JavaScript will remove the element from the document. The syntax for the **removeChild()** method is shown below.

```
Obj.removeChild("element-name")
```

JavaScript setAttribute()

The **setAttribute()** method is used to set or add an attribute to a particular element and provides a value to it. If the attribute already exists, it only set or changes the value of the attribute. So, we can also use the **setAttribute()** method to update the existing attribute's value. If the corresponding attribute does not exist, it will create a new attribute with the specified name and value. This method does not return any value. The attribute name automatically converts into lowercase when we use it on an HTML element.

Syntax

1. element.setAttribute(attributeName, attributeValue)

attributeName: It is the name of the attribute that we want to add to an element. It cannot be left empty; i.e., it is not optional.

attributeValue: It is the value of the attribute that we are adding to an element. It is also not an optional value.

To get the value of an attribute, we can use the **getAttribute()** method, and to remove a specific attribute from an element, we can use the **removeAtrribute()** method.

removeAttribute()

The **removeAttribute()** method removes the attribute with the specified name.

Syntax

1. element.removeAttribute(attributename)

Parameter Values

attributename: It is the required parameter that specifies the attribute's name to remove from the element. If the attribute doesn't exist, the method doesn't create any error.

getAttribute()

The **getAttribute()** method is used to get the value of an attribute of the particular element. If the attribute exists, it returns the string representing the value of the corresponding attribute. If the corresponding attribute does not exist, it will return an empty string or null.

Syntax

1. element.getAttribute(attributename)

attributename: It is the required parameter. It is the name of the attribute we want to get the value from.

JavaScript Event Handling

JavaScript addEventListener()

The **addEventListener()** is an inbuilt function in JavaScript which takes the event to listen for, and a second argument to be called whenever the described event gets fired. Any number of event handlers can be added to a single element without overwriting existing event handlers.

- **Event**: An action performed by the user on the webpage is known as an event

Syntax:
```
element.addEventListener(event, listener, useCapture);
```

- **event:** event can be any valid JavaScript event. Events are used without "on" prefixes like using "click" instead of "onclick" or "mousedown" instead of "onmousedown".
- **listener(handler function):** It can be a JavaScript function that responds to the event occurring.
- **useCapture:** It is an optional parameter used to control event propagation. A boolean value is passed where "*true*" denotes the capturing phase and "*false*" denotes the bubbling phase.

1. **Click**
2. **Mouseover**
3. **Mouseout**
4. **Doubleclick**
5. **submit**

What is Event Bubbling?

Event bubbling, as I mentioned above, is the phase where the event bubbles up from the target element all the way to the global window object.

How event bubbling works

The event starts to bubble up through the `DOM` hierarchy after it has been processed at the target element. The event then travels from the target element to the root of the document. It passes through all ancestor elements according to how they are nested in the bubbling phase.

The bubbling phase

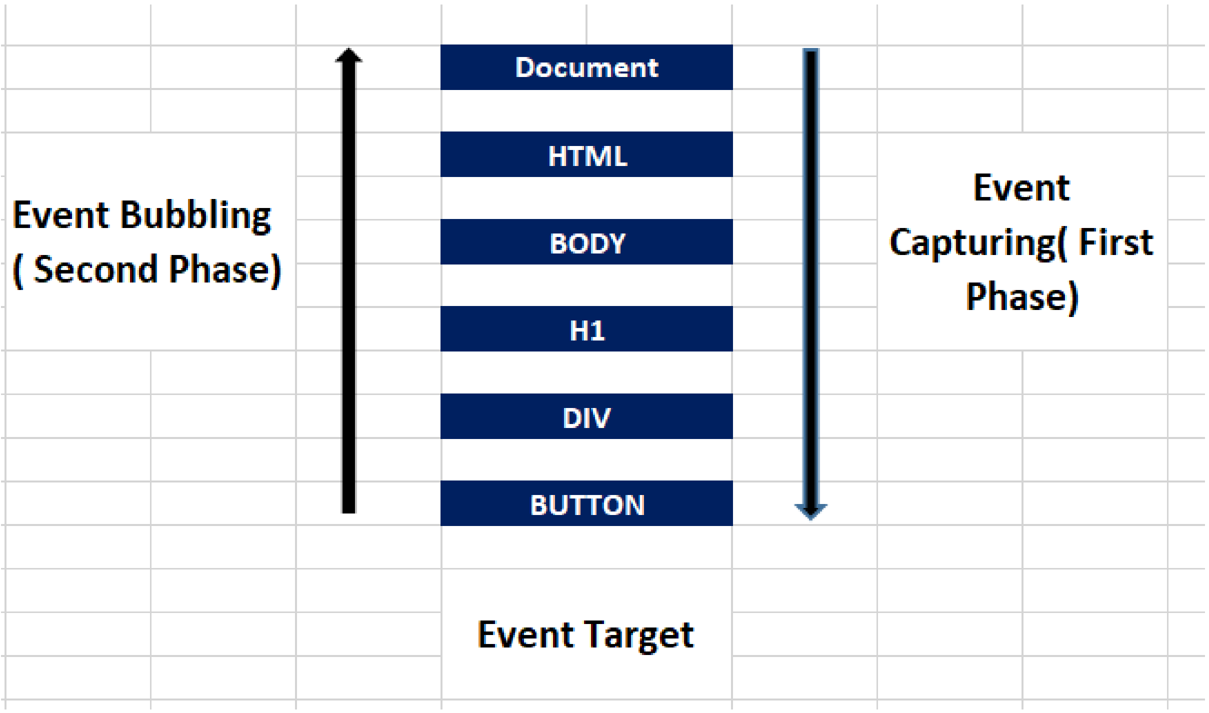
The **bubbling phase**, which is the last phase, is the reverse of the capturing phase. In this phase, the event bubbles up the target element through its parent element, the ancestor, to the global window object. By default, all events you add with `addEventListener` are in the bubble phase.

The capturing phase

The first phase is the **capturing phase**, which occurs when an element nested in various elements gets clicked. Right before the click reaches its final destination, the click event of each of its parent elements must be triggered. This phase trickles down from the top of the DOM tree to the target element.

The target phase

The **target phase** is the second phase that begins immediately after the capturing phase ends. This phase is basically the end of the capturing and the beginning of the bubbling phase.



What is Event Capturing?

Event capturing occurs when a nested element gets clicked. The click event of its parent elements must be triggered before the click of the nested element. This phase trickles down from the top of the `DOM` tree to the target element.

Event capturing can't happen until the third argument of `addEventListener` is set to the `Boolean` value of true.

Whenever the third argument of `addEventListener` is set to `true`, event handlers are automatically triggered in the capturing phase. With this, the event handler attached to an ancestor element will be executed first when an event occurs on a nested element within the `DOM` hierarchy.

Time delays

- setTimeOut() and setInterval() are two functions in js allows to schedule tasks to be executed at later time. Used for animations, polling data from server and other asynchronous tasks.
- **setTimeOut() :**
- Runs the code with time delay

Syntax : window.setTimeOut(function , delay) ;

- **function**: The function you want to execute.
- **delay**: The amount of time (in milliseconds) before the function is executed

- **setInterval() :**
- The function similar to setTimeOut, but it schedules a function to be executed repeatedly at specified interval.

  Syntax : window.setInterval(function , delay) ;

- **function**: The function you want to execute.
- **delay**: The amount of time (in milliseconds) before the function is executed

- cancel a **setTimeout**, you use **clearTimeout**
- cancel a **setTimeout**, you use **clearTimeout**

JavaScript Promise

JavaScript Promise are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.

Prior to promises events and callback functions were used but they had limited functionalities and created unmanageable code. Multiple callback functions would create **callback hell** that leads to unmanageable code. **Promises** are used to handle asynchronous operations in **JavaScript**.

Syntax:

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

Parameters

- The promise constructor takes only one argument which is a callback function
- The callback function takes two arguments, *resolve* and *reject*
    - Perform operations inside the callback function and if everything went well then call resolve.
    - If desired operations do not go well then call reject.

A Promise has four states:

1. **pending**: Promise is still pending i.e. not fulfilled or rejected yet

2.fulfilled: Action related to the promise succeeded

3.rejected: Action related to the promise failed

4.settled: Promise has been fulfilled or rejected

Promise Consumers: Promises can be consumed by registering functions using ***.then*** and ***.catch*** methods.

Promise then() Method: It is invoked when a promise is either resolved or rejected. It may also be defined as a carrier that takes data from promise and further executes it successfully.

Promise catch() Method: It is invoked when a promise is either rejected or some error has occurred in execution. It is used as an Error Handler whenever at any step there is a chance of getting an error.

Async and Await in JavaScript

Async/await is a feature in JavaScript that allows you to **work with asynchronous code** in a more synchronous-like manner, making it easier to write and understand asynchronous code.
Async Functions always return a promise. **Await Keyword** is used only in Async Functions to wait for promise.

Async Function

The Async function simply allows us to write **promises-based** code as if it were synchronous and it checks that we are not breaking the execution thread.
Async functions will always return a value. It makes sure that a promise is returned and if it is not returned then JavaScript automatically wraps it in a promise which is resolved with its value

Async Syntax

```
async function myFunction() {
  return "Hello";
}
```

Await Keyword

Await is used to wait for the promise. It could be used within the async block only.
It makes the code wait until the promise returns a result.

Await Syntax:

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