UI FULL STACK WEB DEVELOPMENT

- What is Full Stack?

Full Stack Web Developer

A full stack web developer is a person who can develop both client and server software.

- In addition to mastering HTML and CSS, he/she also knows how to:

    1. Program a browser (like using JavaScript, jQuery, Angular, or Vue)
    2. Program a server (like using PHP, ASP, Python, or Node)
    3. Program a database (like using SQL, SQLite, or Mongo DB)

Client Software (Front End): HTML5, CSS3, BOOTSTRAP5, JAVA SCRIPTS, ADV JAVA SCRIPTS.

- Server Software (Back End) : NODE JS, EXPRESS JS , MANGO DB

Front End:

- HTML4&5
- CSS2&3
- JAVA SCRIPTS
- ADVANCED JAVA SCRIPTS
- JQUERY
- ECMA 6
- BOOTSTRAP 5

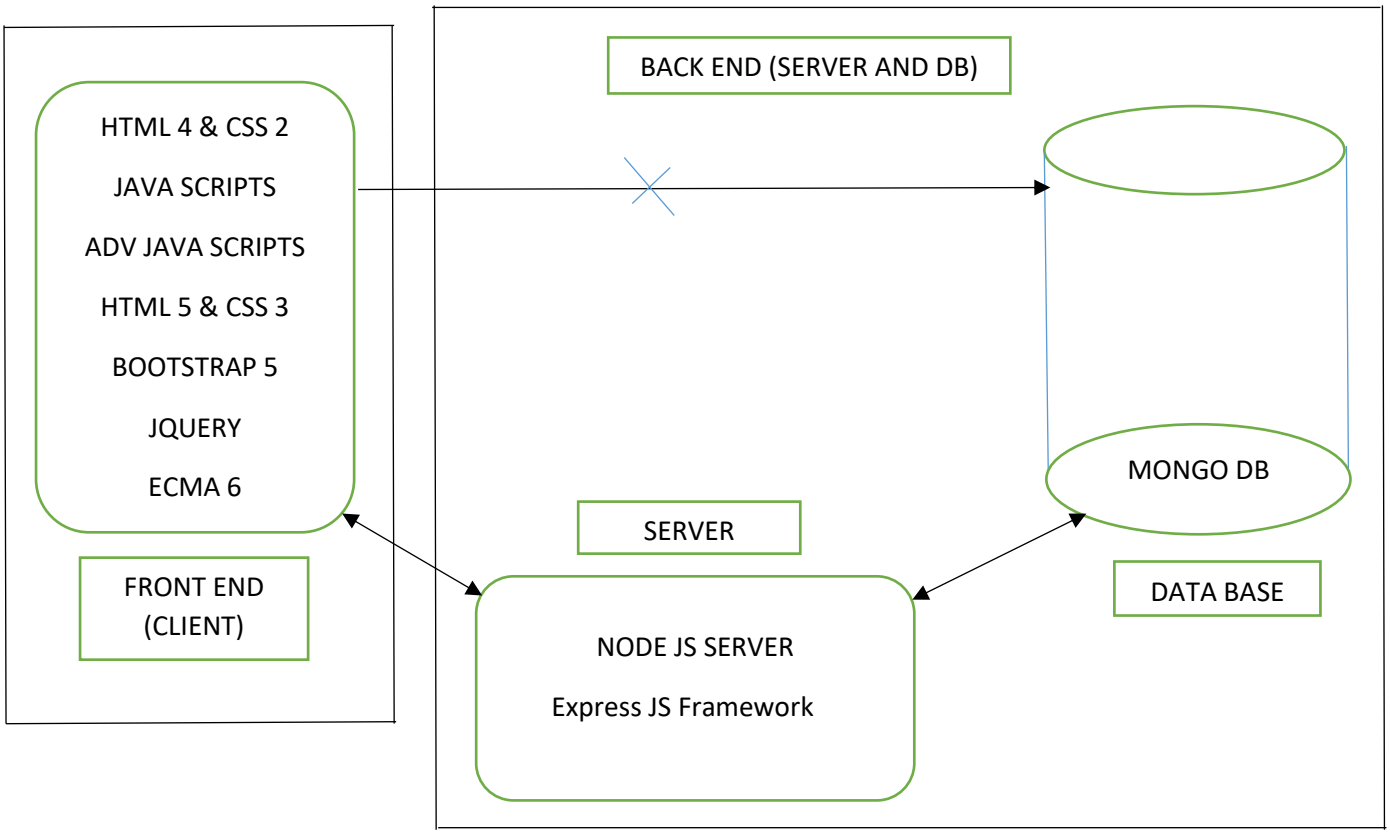
BACK END:

- NODE JS
- EXPRESS JS
- SOCKET IO

DATA BASE:

- MANGO DB.

- **Front End & Back End Technologies:**



Front End:

It only defines UI which a user or a client will unable to access the data in the database.

- **HTML:**

  HTML Stands for **Hyper Text Markup Language** which is capable for display the data within the page means the user/client were unable to access the Data from the Data Base (DB). It is only to display what it has.

- **CSS:**

  CSS Stands for **Cascading Style Sheets**. In order to add beautification to the HTML Page.

- **Static Web Pages:**

  A **static web page** (sometimes called a **flat page** or a **stationary page**) is a web page that is delivered to the user's web browser exactly as stored.

- **Dynamic Web Pages:**

  A **Dynamic Website** (also referred to as a database-driven site) requires web programming and database design. A **dynamic website** contains

information and content that changes, depending on factors such as the viewer of the site, the time of the day, the time zone, or the native language of the country the viewer).

- **Java Scripts:**

  JavaScript is a text-based programming language used both on the client-side and server-side that allows you to make web pages interactive. Where **HTML** and **CSS** are languages that give structure and style to web pages, **JavaScript** gives web pages interactive elements that engage a user.

- **ECMA 6:**

  ES6 stands for **ECMAScript 6**. **ECMAScript** was created to standardize **JavaScript**, and **ES6** is the **6th** version of **ECMAScript**, it was published in 2015, and is also known as **ECMAScript** 2015.

- **BootStrap:**

  Bootstrap is the most popular CSS Framework for developing **responsive** and mobile-first Websites.

- **JQuery:**

  JQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API (Application Programming Interface) that works across a multitude of browsers.

- **Back End:**

Back end Development refers to the server side of **development** where you are primarily focused on how the site works. ... This type of **web development** usually consists of three parts: a server, an application, and a database. Code written by **back end** developers is what communicates the database information to the browser.

- **Node.JS:**

Node. **Js** is a JavaScript **run**-time environment built on Chrome's V8 JavaScript engine. It comes with an http module that provides a set of functions and classes for building a HTTP **server**. For this basic HTTP **server**, we will also be using file system, path and URL, all of which are native **Node.**

- **Express.JS:**

  Express. **Js** is a free and open-source web application framework for Node. **Js**. It is **used** for designing and building web applications quickly and easily.

Data Base (DB):

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). ... The data can then be easily accessed, managed, modified, updated, controlled, and organized.

- **Mongo DB :**

Mongo DB is a document database with the scalability and flexibility that you want with the querying and indexing that you need.

Software we are going to use:

- **Browsers:** Edge, Internet Explorer, Google Chrome, Safari, Firefox etc.…
- **Editors :**Sublime Text, Visual Studio Code(Microsoft's)
- **Repository:** GitHub
- **Tools:** Node.JS (This tool is mostly used by **programmers** who use JavaScript to write Server-Side scripts.)
- **Data Base:** Mongo DB & Mongo DB Compass.

- **What is Mark-up Language?**

Language that uses tags to represent the content. Tag is in the form of <Tag Name >

HTML (Hypertext Mark-up Language)

- **What is HTML:**

  HTML Stands for **Hyper Text Markup Language** which is capable for display the data within the page means the user/client were unable to access the Data from the Data Base (DB). It is only to display what it has.

- **Structure of HTML :**

```
<!Doctype HTML>
<html>
    <head>
        <title>
        </title>
    </head>
  <body>
  </body>
</html>
```

- **Creation of Sample Web Page Using HTML;**

```
<html>
<head>
<title> Sample Web Page </title>
</head>
<body>
<h1>Hello World <h1>
</body>
</html>
```

Save this file as .HTML or with .HTM Extension.

- A predefined language using which we could able to add content and resources to the Web pages. It comes with Predefined set of HTML Tags used to add different types of content accordingly.

Some Examples of Tags are:

- <HTML>
  It is used to hold complete content of the Web Page.
- <head>
  It is used to indicate the head of the Web Page.
- <title>
  It holds the title of the Web Page
- <body>
  To hold the actual content of the page

- <p>
  Paragraph tag used to hold the multiline text content
-
  Break tag adds single line break
-
  Hold block content
-
  Inline elements to hold content in the same line
- <table>
  To render the content in the row and column way
-
  To render the content in unordered or ordered way

-
  To display the list of items

➢ **HTML Attributes;**

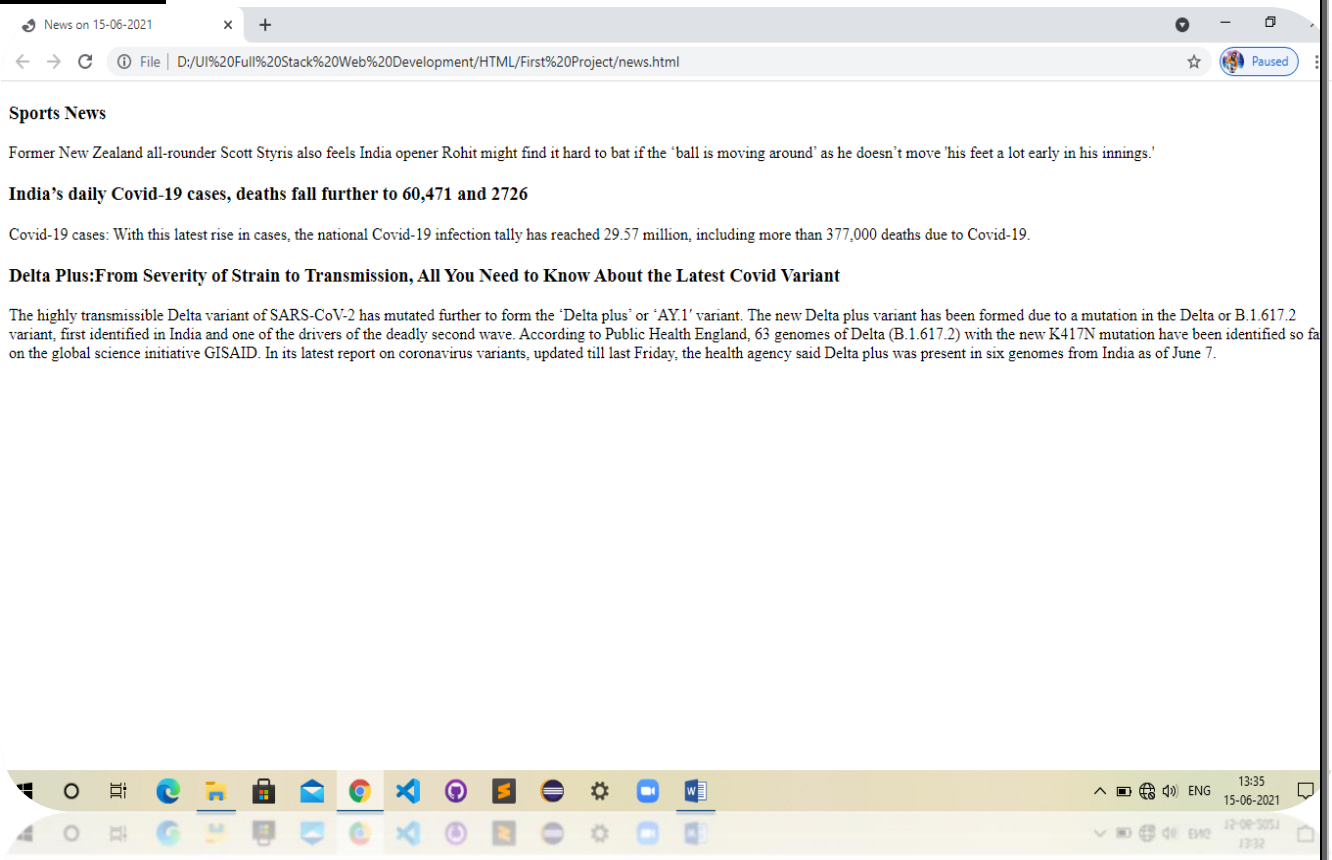
We can add extra information to the HTML elements through HTML Attributes. The attributes can be both predefined and user defined. Following some of the predefined attributes can be added to elements,

- id: using which we can unique reference to elements.
- Name: name value can be added to elements.
- Class: using which we can add single/multiple CSS classes to elements.
- Style: to add single or multiline CSS properties to the elements.
- Alt: using which we can add alternative text content.
- Title: using which we can add title to any HTML elements.

➢ Sample Web Page Using HTML Attributes:

```
<html>
    <head>
       <meta charset="utf-8">
       <title>News on 15-06-2021</title>
    </head>
    <body>
    <h3> Sports News </h3>
    <p id="sportsNews" name="cricket" title=" about ICC World Test Champion Ship 2021 india VS newzeland">
    Former New Zealand all-rounder Scott Styris also feels India opener Rohit might find it hard to bat if the
    'ball is moving around' as he doesn't move 'his feet a lot early in his innings.'</p>
    <h3>India's daily Covid-19 cases, deaths fall further to 60,471 and 2726 </h3>
    <p id="Covid-19" name="India'scovidnews" title="Covid-19 cases in india as on 15-06-2021">Covid-19 cases:
    With this latest rise in cases, the national Covid-19 infection tally has reached 29.57 million, including
    more than 377,000 deaths due to Covid-19.</p>
    <h3>Delta Plus:From Severity of Strain to Transmission, All You Need to Know About the Latest Covid Variant
    </h3>
    <p id="CoronaVirus" name="covid news" title="Delta Plus variant From Severity of Strain to Transmission,
    All You Need to Know About the Latest Covid Variant">The highly transmissible Delta variant of SARS-CoV-2
    has mutated further to form the 'Delta plus' or 'AY.1' variant. The new Delta plus variant has been formed
    due to a mutation in the Delta or B.1.617.2 variant, first identified in India and one of the drivers of
    the deadly second wave. According to Public Health England, 63 genomes of Delta (B.1.617.2) with the new
    K417N mutation have been identified so far on the global science initiative GISAID. In its latest report on
    coronavirus variants, updated till last Friday, the health agency said Delta plus was present in six
    genomes from India as of June 7.</p>

    </body>
</html>
```

OUTPUT:



*CSS (*Cascading Style Sheets)

- **Inline CSS:**

  An inline CSS is used to apply a unique style to a single HTML Element.
  An inline CSS uses the **STYLE** attribute of an HTML Element.

  - **STYLE Attribute:**

    Setting the style of an HTML element, can be done with the style attribute.
    The HTML STYLE attribute has the following Syntax,

```
<TagName style=" ">
        ……
</TagName>
```

    Or

```
<style>
.className{
      ……..
}
</style>
```

    Style is a predefined tag capable of holding any no.of CSS Classes.

  - **Background:**

    The CSS Background indicates the background color of a HTML page.

  - **Color:**

    The CSS Color indicates the Text Color of a HTML Page.

- **CSS Class:**

  A CSS Classes is a group of set of CSS properties and assigning a name to it.

  - A concept of grouping all the required CSS Properties as an individual blocks, assigning a user defined name to it is called as a **CSS Class.**
  - Every CSS Class name should be started with **. (Dot)** operator.
  - In a single page we can define any number of CSS classes.
  - Class is a predefined attribute using which we could able to inject one or multiple CSS Classes to a single HTML Element.
  - Once a CSS Class been defined it can be used any no.of.times to any no.of.elements.

  Syntax:

```
.className {
        ……..// Set of CSS Properties
    }
```

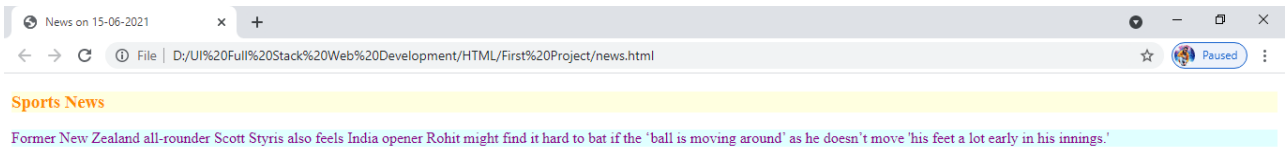
Classes are used by CSS is to select and access a specific elements.

- **Sample Webpage using Style Attribute:**

CODE:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title >News on 15-06-2021</title>
            <style type="text/css">
                .newsHeading {
                    background: lightyellow;
                    color: darkorange;
                }

                .newsDescription {
                    background: lightcyan;
                    color: darkmagenta;
                }

            </style>
    </head>
    <body>
        <h3 class="newsHeading"> Sports News </h3>
            <p id="sportsNews" name="cricket" title=" about ICC
            World Test Champion Ship 2021 India VS newzeland"
            class="newsDescription">Former New Zealand all-
            rounder Scott Styris also feels India opener Rohit might find
            it hard to bat if the ‘ball is moving around’ as he doesn’t
            move 'his feet a lot early in his innings.'</p>

    </body>

</html>
```

OUTPUT:



➢ CSS Color Values:

In CSS, colors can also be specified using RGB values.

Ex: background-color : rgb (255, 99, 71);

➢ 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.

Ex: <a href:https://www.google.com>press here

➢ Image Tag:

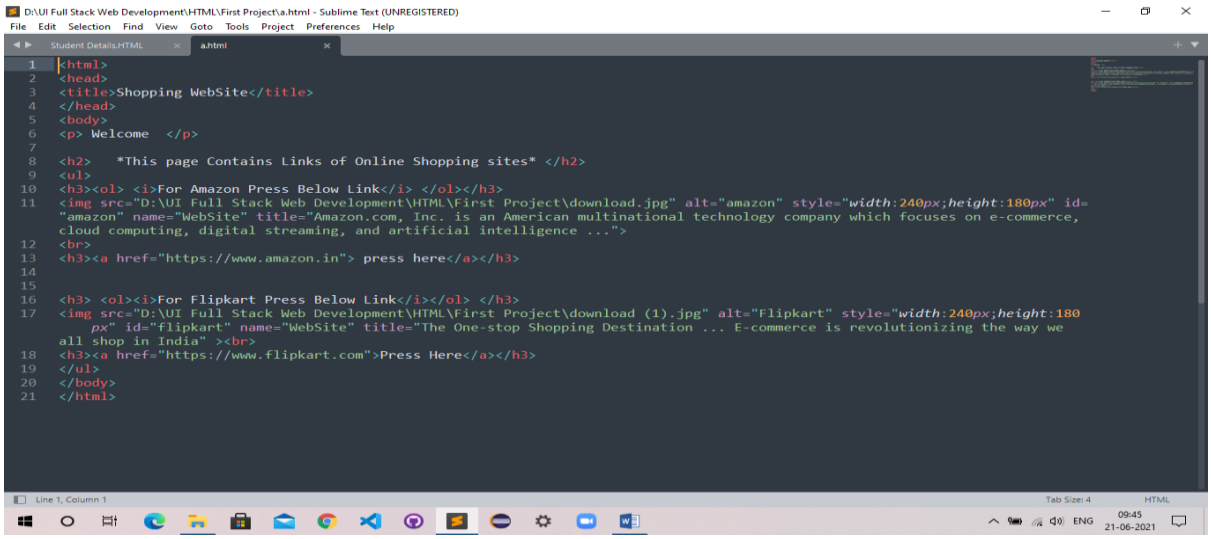
The **<img>** tag is used to embed an image in an HTML page.

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.

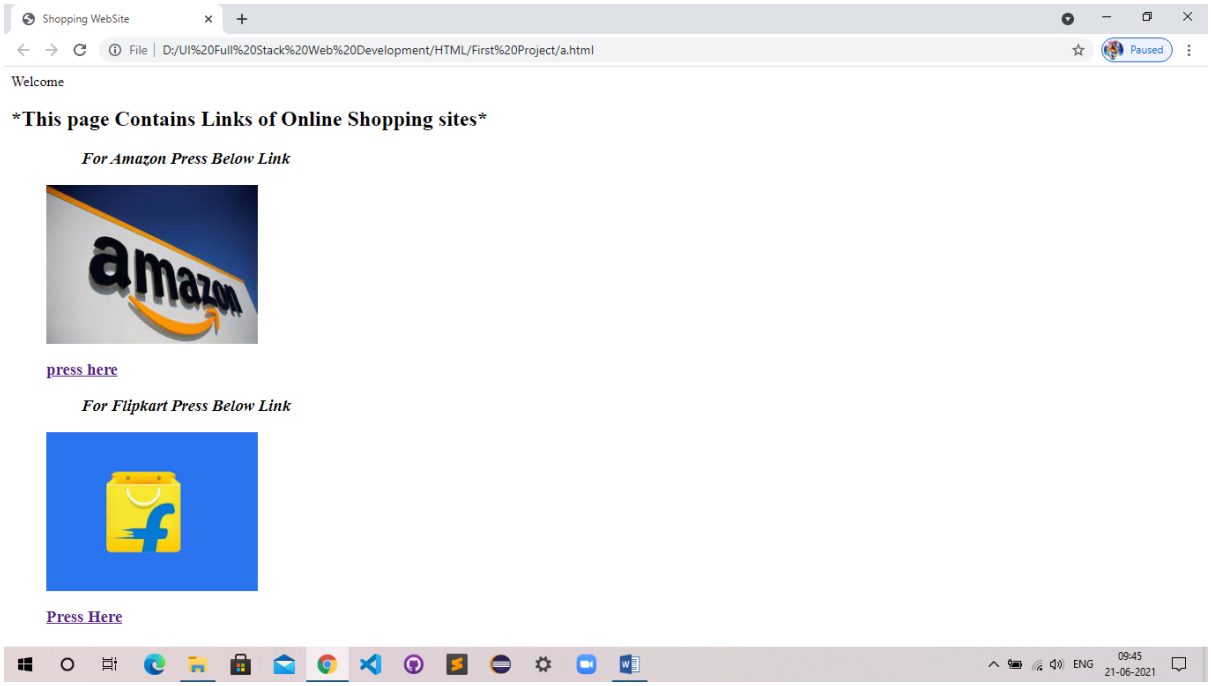
Ex: <img src="img_213392.jpg" width="100px" height="500px">

<IMG> tag is a self-closing tag.

➢ Sample web page using image tag and anchor tag:



OUTPUT:



- **Target Attribute:**
  - The **target** attribute specifies a name or a keyword that indicates where to display the response that is received after submitting the form.
  - Ex:

```
<h3><a href="https://www.amazon.in" target="_blank"> press here</a></h3>
```

- **Different ways to inject CSS to an Element:**

1) Adding inline CSS through **<style>** attribute.
   Ex :< p style=”color: red;” > abcd </p>
2) Creating **Class** and injecting through Class attribute.
   Ex : <style type="text/css">

```
                .newsHeading {
                    background: lightyellow;
                    color: darkorange;
                }
    </style>
<body>
        <h3 class=”newsHeading”> aaaaa </h3>
</body>
```

3) Creating a Class through **ID** of an element.

```
Ex : #idname {
            background: lightyellow;
            color: darkorange;
      }
<body>
   <p id="idname"> Ananth </p>
```

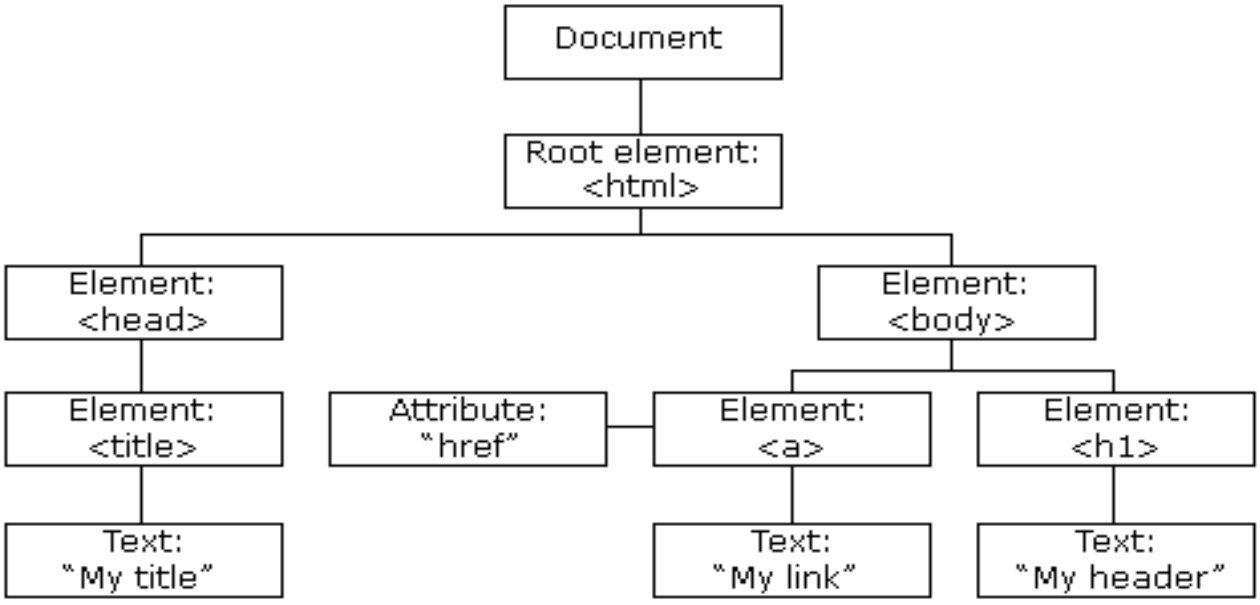
4) Creating a CSS class using the **Tag Name** itself.

```
Ex : h3 {
        background: lightyellow;
        color: darkorange;
      }
<body>
   <h3> Hi Ananth </h3>
```

- **DOM (Document Object Model) Structure :**

For every web page browser dynamically creates a DOM Structure, which indicates the tree structure of the current web pages with all the elements.

DOM Structure specifies the relationship between the Elements, list of Elements participated within the page, attributes and its corresponding values etc...



The above diagram is example HTML Tree structure.

➢ Debugger Tool:

- Every browser by default comes with a default debugger tool using which we could able to debug any web page running under the Browser.
- It shows DOM Structure of the current Webpage.
- It provides an option to create a dynamic HTML Elements while the page is running.
- It provides an option to edit, delete or update any HTML Element and its corresponding attributes while the page is running.
- Any CSS property of an HTML Element can be edited, deleted, add a new property can be done through debugging tool.
- It also provides an option to explore the list of resources been loaded in the current Webpage.
- It also provides an option to explore the list of resources getting loaded through Network or Server (image, JS files, CSS files, XHR request etc…).
- Press ctrl + shift + I in a web page in browser to get debugging tool

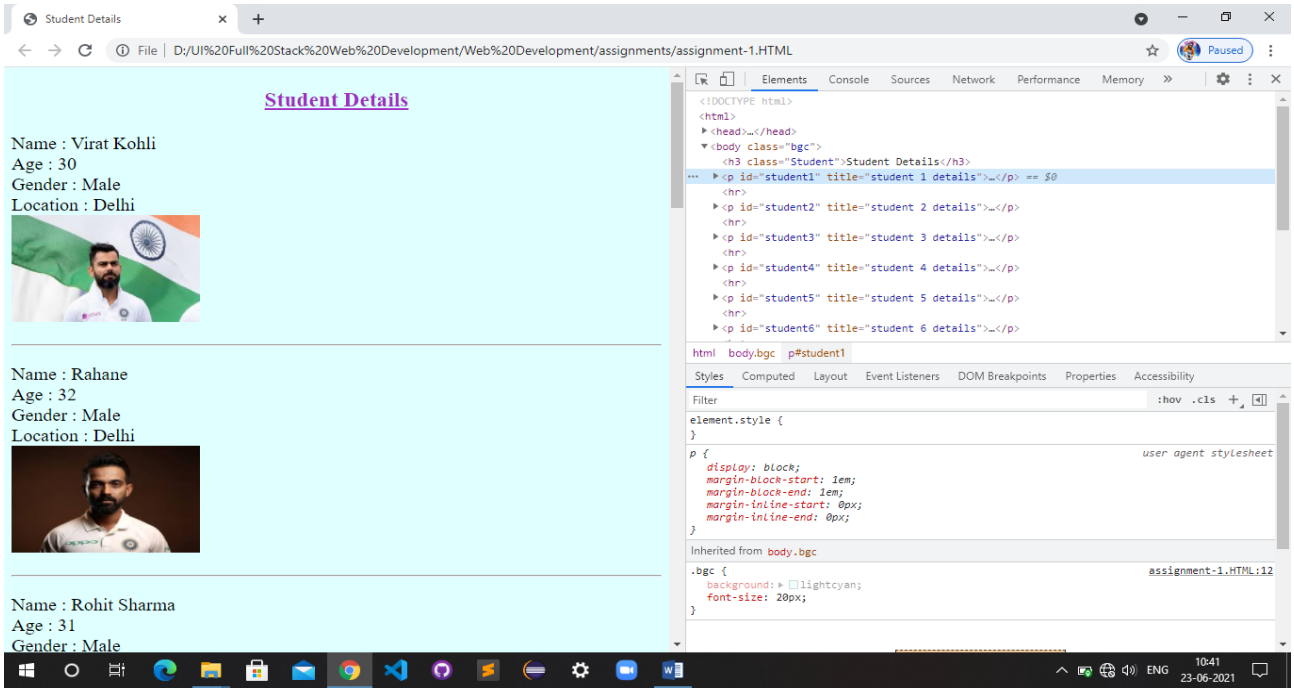


Fig. The above picture shows the debugging tool of the running webpage.

➢ Priority chain of CSS:

As we have the four different ways to apply the CSS Properties for the HTML Element, if the same CSS Property been applied to a single element using all the four different ways with different values, by default Browser follows the below priority order,

- Among all the four different ways CSS been applied inline (through **STYLE** attribute) will always takes the higher priority in order.
- CSS been applied through **ID**, which takes the second priority order.
- CSS been applied through **Class**, which takes third priority in order.
- CSS been applied through **TAG**, which takes least priority in order.

Ex:

```
<head>
    <style type="text/css">
        .newsHeading {
            background: lightyellow;
            color: darkorange;
        }
        .newsDescription {
            background: lightcyan;
            color: darkmagenta;
        }
        #sportsNews {
            background: deeppink;
            color: goldenrod;
        }
        p {
            background: blueviolet;
        }
    </style>
</head>
<body>
    <h3 class="newsHeading"> Sports News </h3>
    <p id="sportsNews" name="cricket" style="background: darkmagenta; font-size:
    30px;" class="newsDescription">Former New Zealand all-rounder Scott Styris also
    feels India opener Rohit might find it hard to bat if the ‘ball is moving around’ as he
    doesn’t move 'his feet a lot early in his innings.'</p> </body>
```

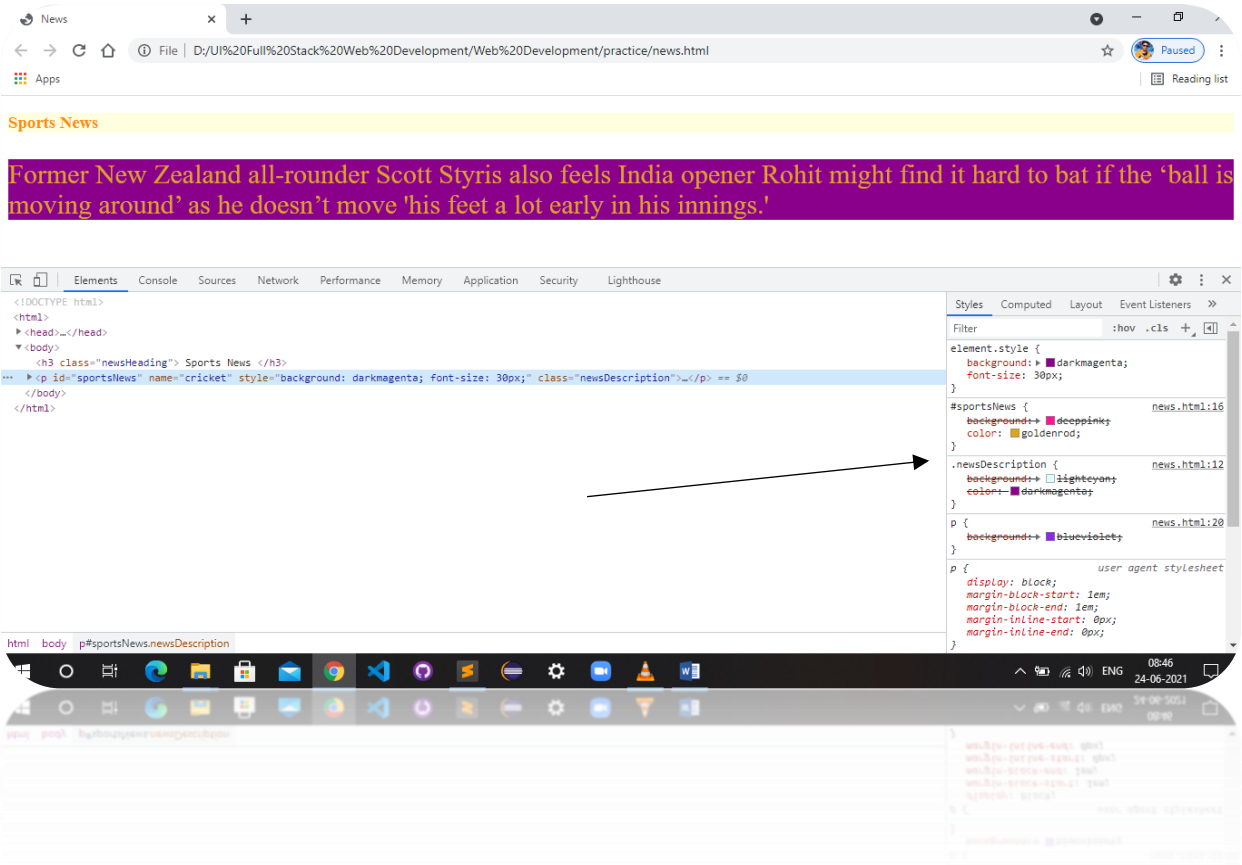


Fig: The above picture represents the CSS priority

❖ **Note:**

Irrespective of the above priority order any CSS property being added with '! Important' will always takes the Higher Priority.

Ex:
```
<head>
    <style type="text/css">
        .newsHeading {
            background: lightyellow;
            color: darkorange;
        }
        .newsDescription {
            background: lightcyan;
            color: darkmagenta;
        }
        #sportsNews {
            background: deeppink !important;
            color: goldenrod;
        }
        p {
            background: blueviolet;
        }
    </style>
</head>
```

```
<body>
        <h3 class="newsHeading"> Sports News </h3>
        <p id="sportsNews" name="cricket" style="background: darkmagenta; font-size:
        30px;" class="newsDescription">Former New Zealand all-rounder Scott Styris also
        feels India opener Rohit might find it hard to bat if the 'ball is moving around' as he
        doesn't move 'his feet a lot early in his innings.'</p>

</body>
```

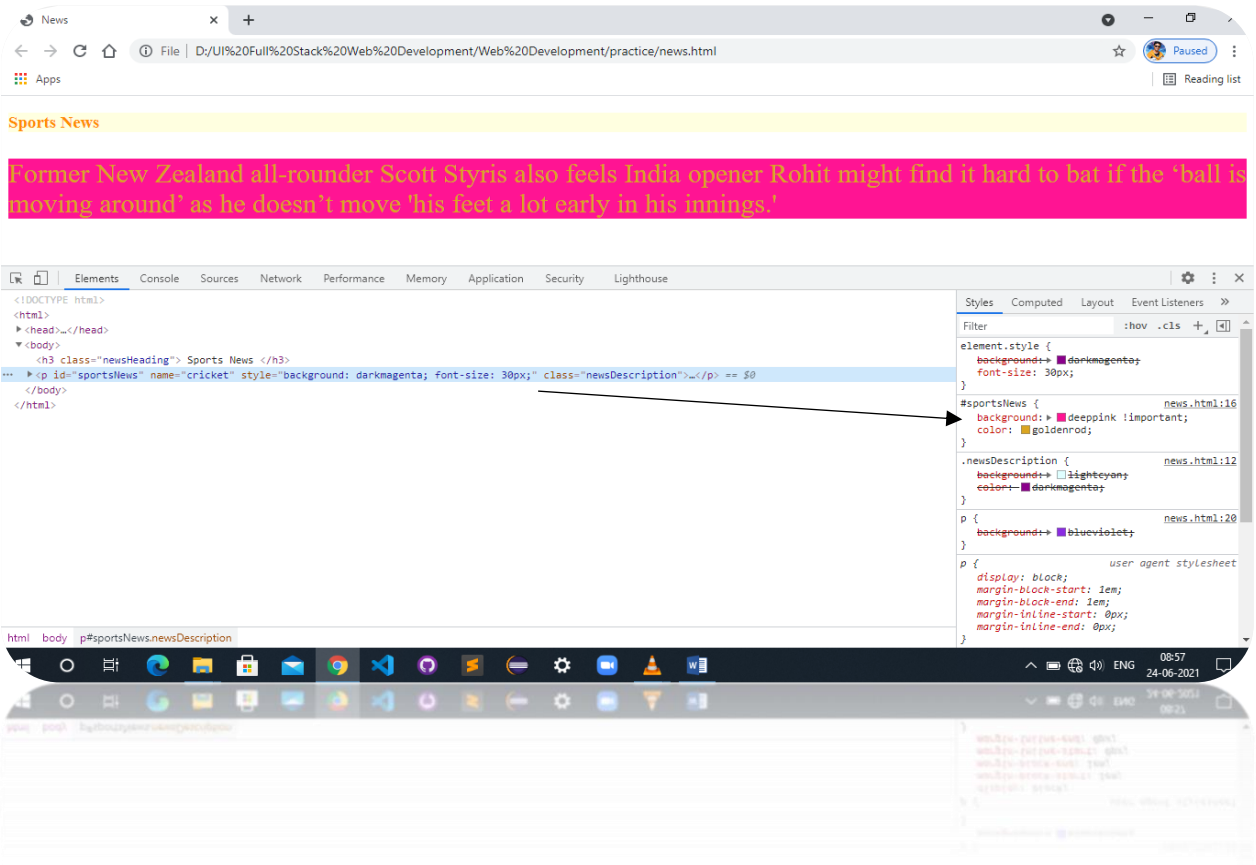


Fig: Example with **!important**

- **<u>Inline and Block level Elements:</u>**

  All the HTML Elements are been categorized into two types

  I. Block Level Element
  II. Inline Element.

  - **<u>Block level Elements:</u>**

    Any HTML Element which comes under the block level category holds the following properties,

    - It occupies 100% width of its container by default.
    - Even though it occupies 100% width we can still control the dimensions of the block level element through CSS width and height properties.

- While getting rendered within the page by default it comes to new line and gets rendered.
- The element which is following a block level element also automatically comes to a new line and gets rendered.
- Block level elements are mainly used to hold the group of relative items as like an individual block.
- **'Div'** tag is a best example for a block level element.

Ex: <body>

<h3> example of block level element </h3>

one

two

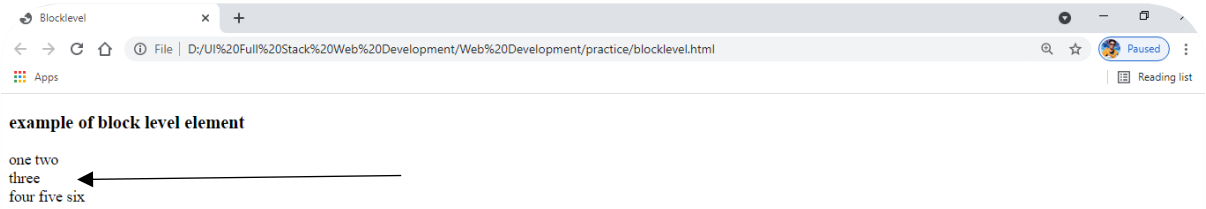
** three **

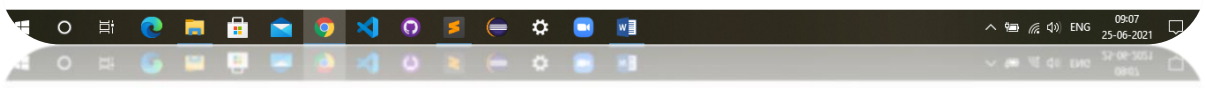
four

five

six

</body>





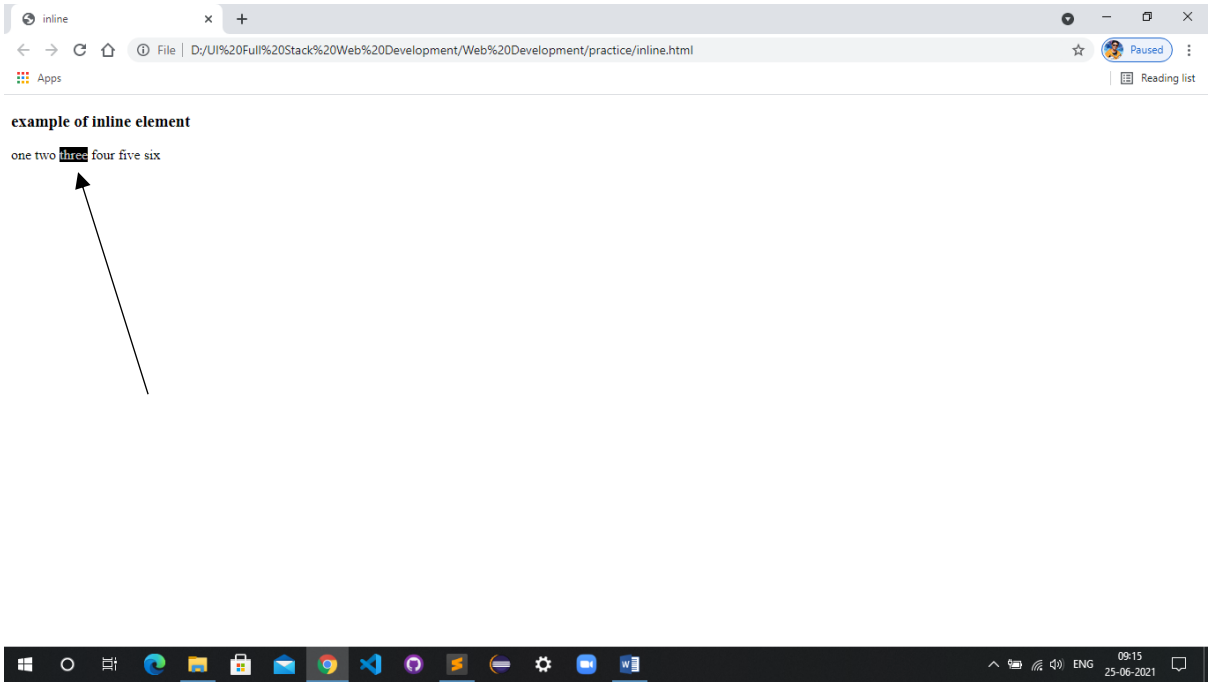
❖ **Inline Element:**

Any HTML Element comes under inline category holds following properties,

- While rendering on the page all the inline elements tries to render within the same line.
- Inline elements always occupies the width within the container based on the content it is holding.
- We cannot control the dimensions of an inline element.
- **'Span'** tag is the best example for an inline element.

Ex:

```
<head>
    <meta charset="utf-8">
    <title>inline</title>
        <style type="text/css">
            .clr {
                color: #ccc;
                font-size: 16px;
                background: black;
            }
        </style>
</head>
<body>
    <h3> example of inline element </h3>
    one
    two
    <span class="clr">three</span>
    four
    five
    six
</body>
```



- **Span Tag:**
  The tag is an inline container used to mark up a part of a text, or a part of a document.
- **Div Tag:**
  The tag defines a division or a section in an HTML document.

➢ 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

➢ CSS 'Padding' and 'margin' properties:

By default while the elements getting rendered on the page they try to occupy very next line of the previous element without maintaining any space.

In order to add space or gap between or within the elements we make use the CSS Properties **'Padding'** and **'Margin'**.

➢ CSS 'Padding' property:

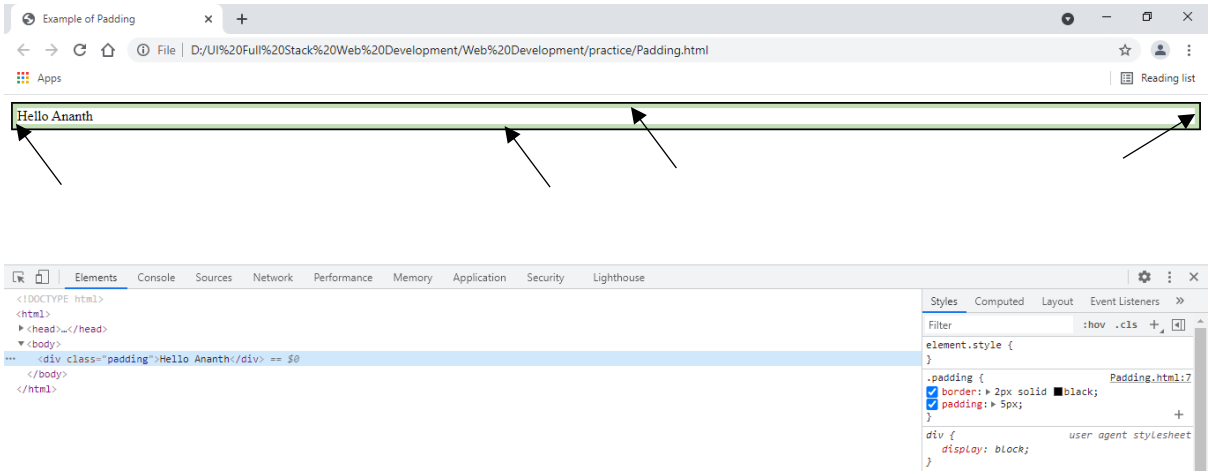
Through **'padding'** property we could able to add space or gap between the border and the content of the container.

Example:

- Padding : 5px;
  {Adds Padding of 5px to all the Direction (Top, Left, Right & Bottom)}
- Padding : 5px 10px; **{Adds Padding of 5px to Top and Bottom and 10px to Left and Right}**
- Padding 5px 10px 15px 20px; **{Adds 5px to Top, 10px to Right, 15px to Bottom, 20px to Left}**
- Padding-top : 10px; **{Adds 10px to Top}**
- Padding-left : 10px; **{Adds 10px to Left}**
- Padding-right : 10px; **{Adds 10px to Right}**
- Padding-bottom : 10px; **{Adds 10px to Bottom}**

Ex:

```
<head>
    <meta charset="utf-8">
    <title>Example of Padding</title>
    <style type="text/css">
        .padding {
            border: 2px solid black;
            padding: 5px;
        }
    </style>
</head>
<body>
    <div class="padding">Hello Ananth</div>
</body>
```



In the above picture we can see the green color border space inside the main container which is indicated by arrows occurred by **padding** property.

➢ CSS 'Margin' Property:

Through which we could able to add space or gap above the border of the container.

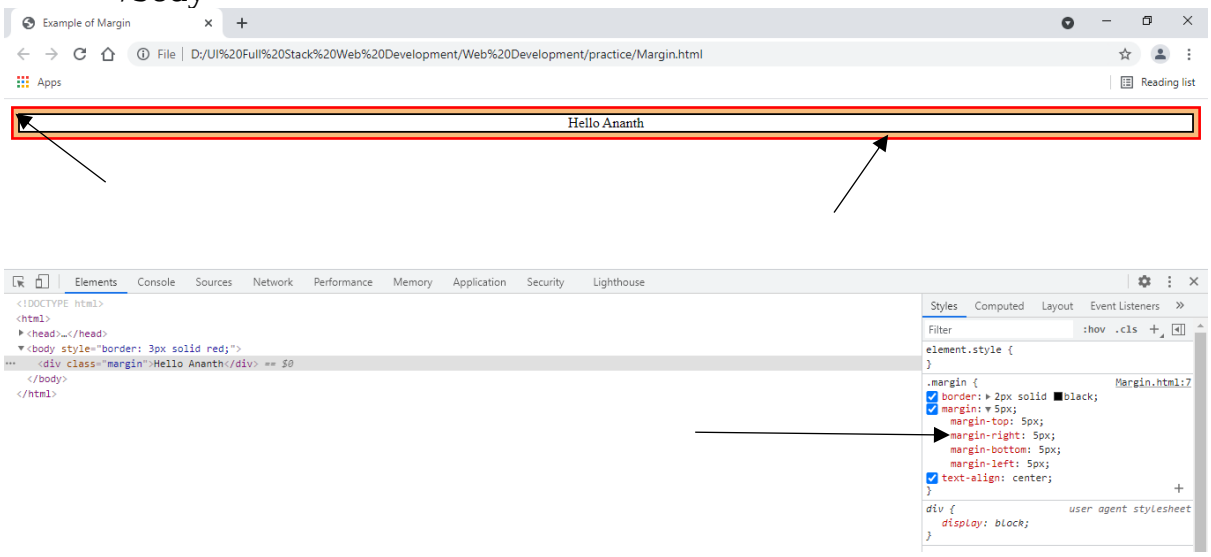
Example:

- Margin : 5px;
  {Adds Margin of 5px to all the Direction (Top, Left, Right & Bottom)}
- Margin : 5px 10px; **{Adds Margin of 5px to Top and Bottom and 10px to Left and Right}**
- Margin 5px 10px 15px 20px; **{Adds 5px to Top, 10px to Right, 15px to Bottom, 20px to Left}**
- Margin-top : 10px; **{Adds 10px to Top}**
- Margin-left : 10px; **{Adds 10px to Left}**
- Margin-right : 10px; **{Adds 10px to Right}**
- Margin-bottom : 10px; **{Adds 10px to Bottom}**

Ex:

```
<head>
    <meta charset="utf-8">
    <title>Example of Margin</title>
    <style type="text/css">
        .margin {
            border: 2px solid black;
            margin: 5px;
            text-align: center;
        }
    </style>
```

```
</head>
<body style="border: 3px solid red;">
        <div class="margin">Hello Ananth</div>
</body>
```



In the above picture we can see the orange color space between two borders which is done by using **Margin** property.

➢ CSS Box Model:

When an element gets rendered on the page, browser calculates the actual dimensions of the existing elements on the page, accordingly it renders the new DOM Element. While calculating the dimensions of an element, it follows Box Model feature.

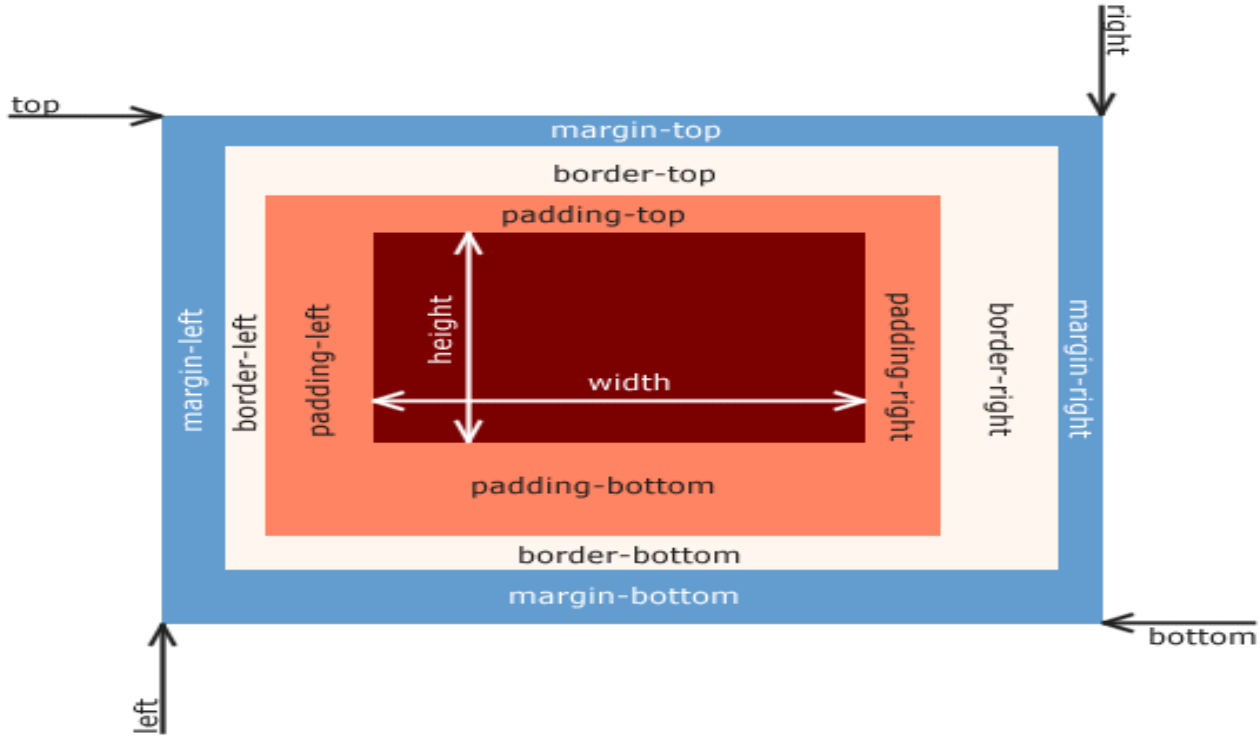
In CSS Box Model property while calculating the actual dimensions of an element, it considers the following CSS Properties,

- The total Border space been Occupied.
- The total Margin space been Occupied.
- Total Padding space been Occupied.
- Actual width & height of the DOM Element.

Explanation of the different parts:

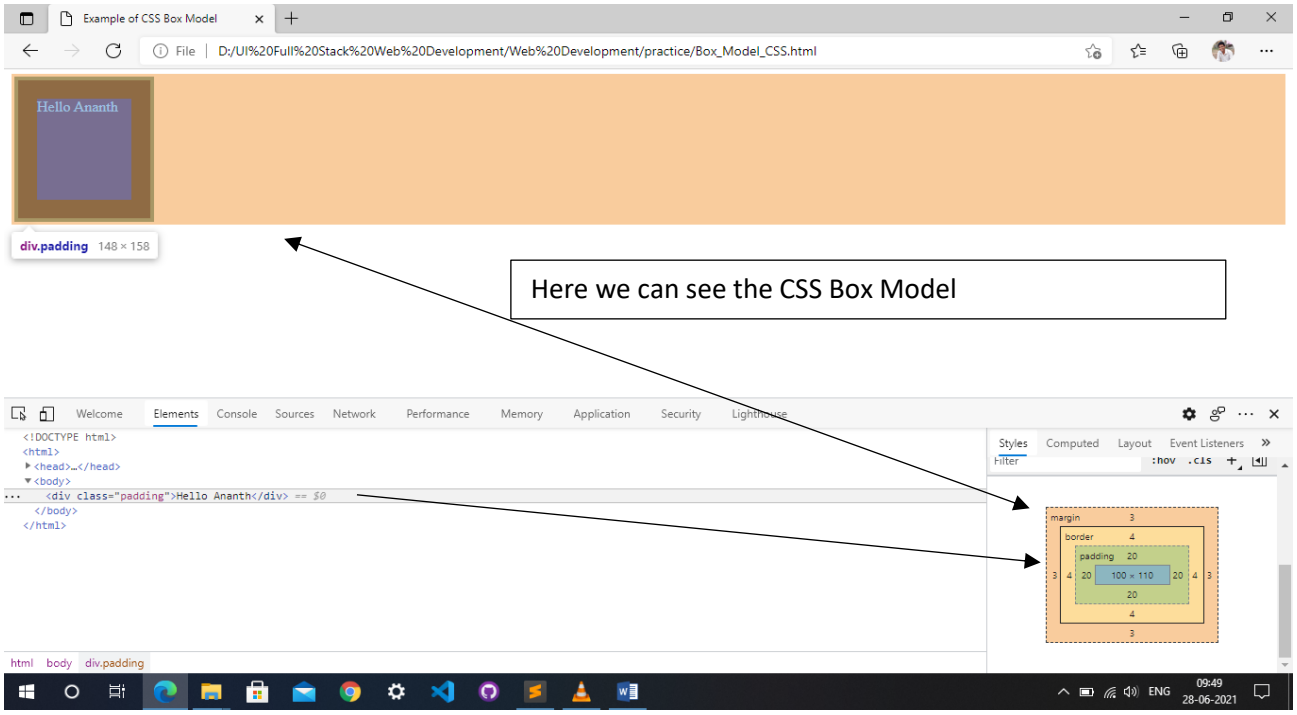
- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content
- **Margin** - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.



Ex: <head>

```
        <meta charset="utf-8">
        <title>Example of CSS Box Model</title>
        <style type="text/css">
            .padding {
                border: 4px solid black;
                padding: 20px;
                margin: 3px;
                height: 110px;
                width: 100px;
                background: darkred;
                color: whitesmoke;
            }
        </style>
    </head>
    <body>
        <div class="padding">Hello Ananth</div>
    </body>
```



➢ HTML UL and OL Tags:

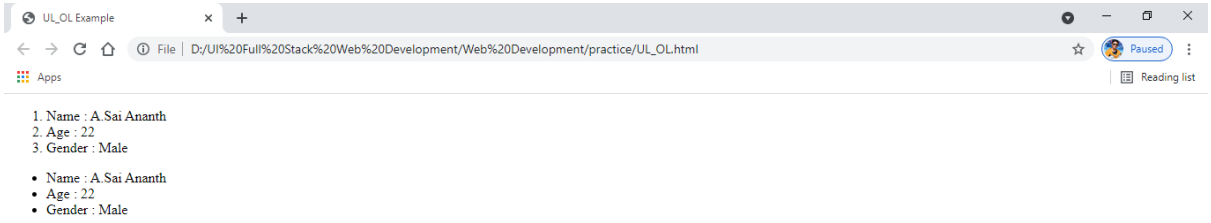
The two predefined HTML tags using which we could able to group relative set of items as an individual block.

'UL' and **'OL'** tags renders the elements almost same where the only difference is **'UL' (Unordered list)** tag renders the element with default bullet symbols, whereas **'OL' (Ordered List)** tag renders the element with default serial number.

- **Syntax:**

```
<ul>
    <li> item-1</li>
    <li> item-2</li>
</ul>
<ol>
    <li> item-1</li>
    <li> item-2</li>
</ol>
```

Ex:
```
<body>
    <div>
        <ol>
            <li>Name : A.Sai Ananth</li>
            <li>Age : 22</li>
            <li>Gender : Male</li>
        </ol>
    </div>
    <div>
        <ul>
            <li>Name : A.Sai Ananth</li>
            <li>Age : 22</li>
            <li>Gender : Male</li>
        </ul>
    </div>
```



- **'list-style'** is the CSS Property through which we can control the type of List style getting rendered for both **'UL'** and **'OL'** tags.
- For More Visit https://www.w3schools.com/cssref/pr_list-style-type.asp

Ex:
```
<head>
    <meta charset="utf-8">
    <title>UL_OL Example</title>
    <style type="text/css">
        .list {
            list-style: decimal;
        }
    </style>
</head>
<body>
    <div>
        <ol>
            <li>Name : A.Sai Ananth</li>
            <li>Age : 22</li>
            <li>Gender : Male</li>
        </ol>
    </div>
```

```
        <div>
            <ul class="list">
                <li>Name : A.Sai Ananth</li>
                <li>Age : 22</li>
                <li>Gender : Male</li>
            </ul>
        </div>
    </body>
```

➢ Adding Background image for Containers:

A background image is specified for almost any HTML Element. To add background image on an HTML Element, use the HTML STYLE attribute and the CSS **background-image** property.

Syntax: background-image : url('Enter the url');

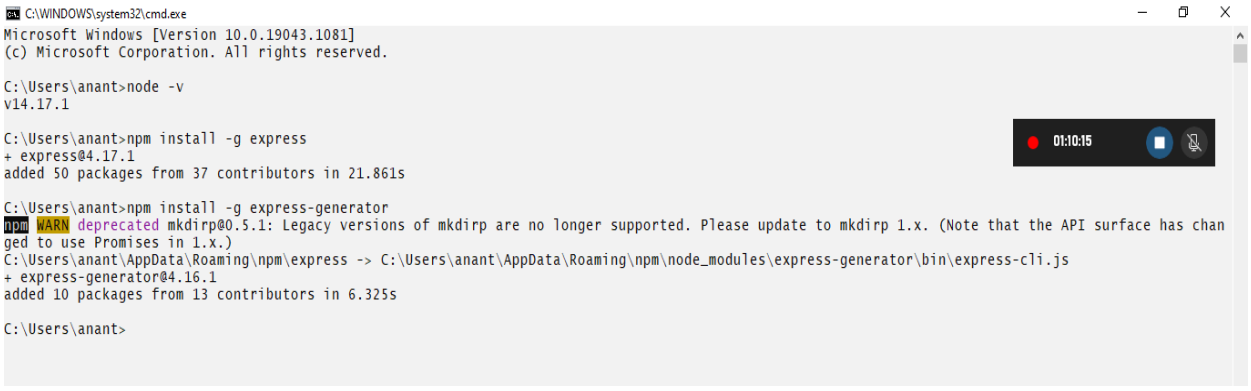
Ex :

```
<head>
    <title>Example of CSS Box Model</title>
    <style type="text/css">
        .padding {
            border: 4px solid black;
            padding: 20px;
            margin: 3px auto;
            height: 230px;
            width: 410px;
            background-image:
            url('https://th.bing.com/th/id/OIP.fknHCHfIA3LC
            nTkqKeA2sQHaEK?pid=ImgDet&rs=1');
            background-repeat: no-repeat;
            color: darkred;
        }
    </style>
</head>
<body>
    <div class="padding">Hello Ananth</div>
</body>
```

- **Background-repeat:** no-repeat; is used for the image not to repeat.

➢ Downloading and Setting Up Node.js Server:

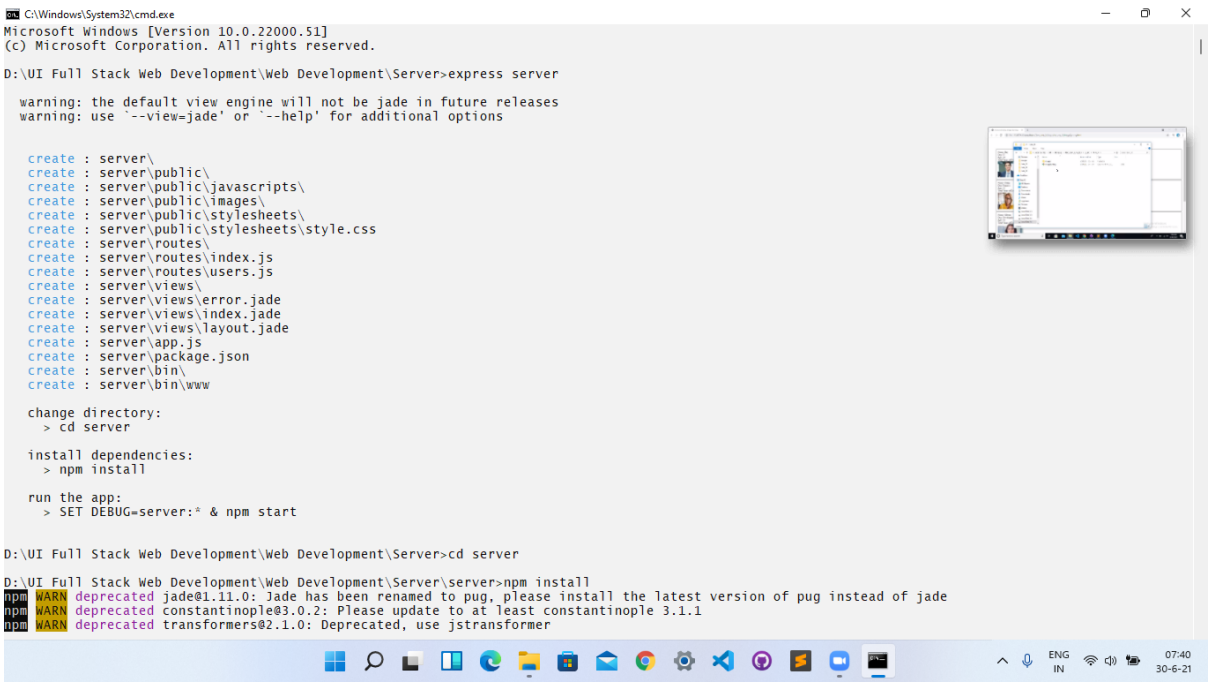
- Open the command prompt and check weather Node.js Server is already installed or not by using command **node –v**.
- If not download it from www.nodejs.org/en/
- After Successful download install the package.
- After Successful installation once again open the command prompt and check the weather the setup is successfully installed or not.
- After type **npm install –g express**. Here npm means Node Package Manager.
- After successful download type the command **npm install –g express-generator**.

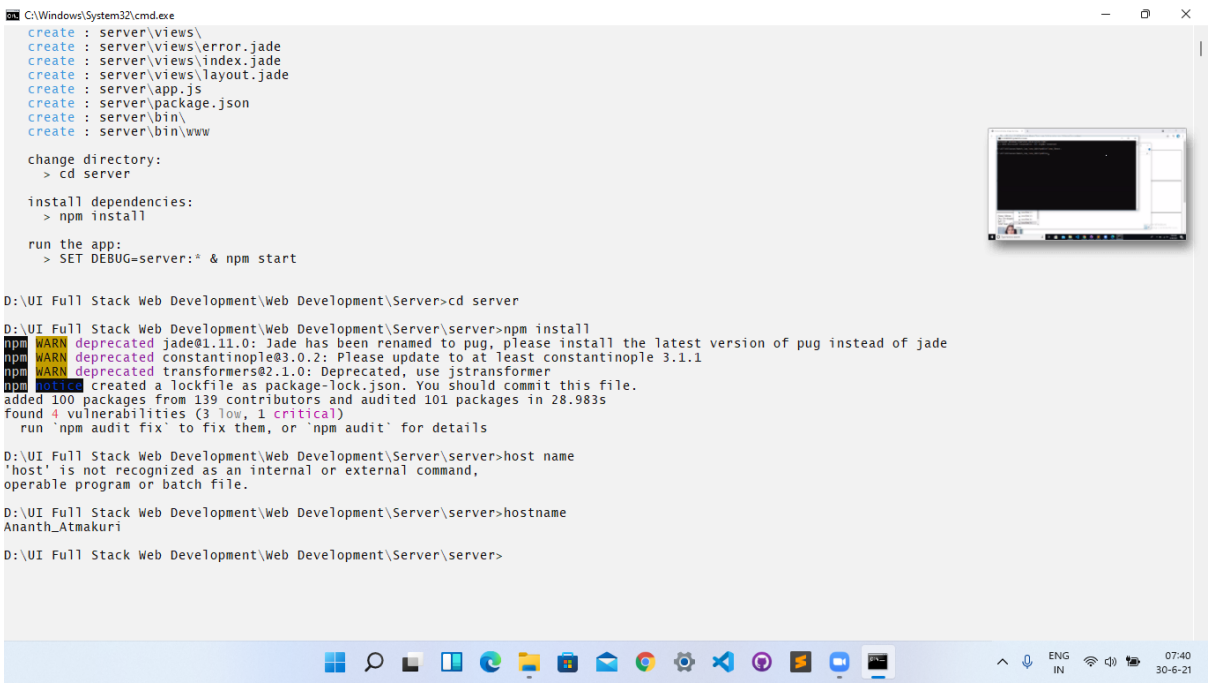


➢ Setting up Node.js Server:

- Open Command Prompt.
- Open the path in the command prompt were you want to setup the server **Ex:** D:\UI Full Stack Web Development\Web Development\Server
- Type **express folder_name** in the next line.

- Go to the folder which you set up the server.
- After type **npm install.**
- After open the app(js) file in editor and go to the line 24 and add
  app.listen(8081, function() {
  console.log("Server is listing at 8081");
  }); // where console.log is used to print the data which is given by us.
- After that check whether it is running or not by using command **npm start**.





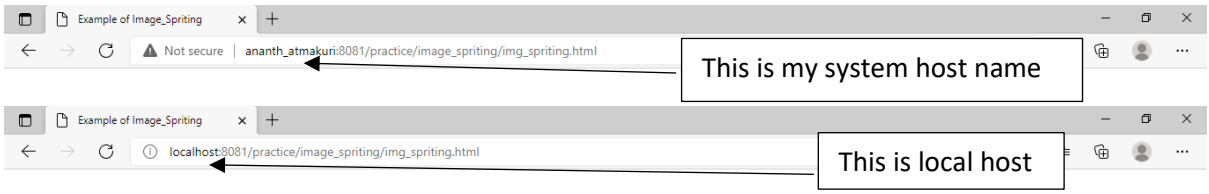
In the above picture you can the server setup.

➢ **Opening the files from the server:**

- Copy the file into the public folder which is present were you created the server.
- Start the server by using command **npm start**.
- Open any file in the public folder where you created.

Ex: http://localhost:8081 /Assignment-2_StudenDetails_Updated.HTML

- We can open the file by using host name and local host



➢ **Image Spriting:**

The concept of merging all the static images to form a single image, through background position property, we show only required image is called as image spriting.

Through image spriting we increase the performance of the page by loading all the static images in a single column.

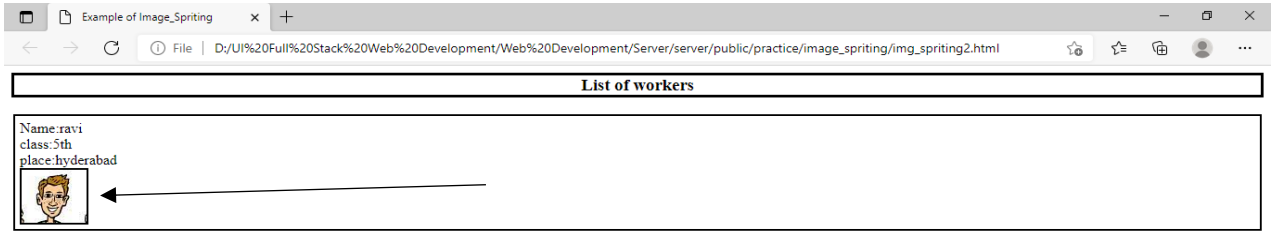
➢ **Note:**

Image spriting is only recommended for static web pages but not for dynamic web pages.

Ex:

```
<html>
    <head>
        <meta charset="utf-8">
        <title>Example of Image_Spriting</title>
        <style type="text/css">
            .mainBorders {
                border: 2px solid;
                padding: 5px;
                margin: 2px;
            }
            .allImages {
                border: 2px solid;
                width: 70px;
                height: 60px;
                background-image: url("images/fullimages.jpg");
            }
            .worker1 {
```

```
                    background-position: 222px -9px;
                }
            </style>
        </head>
        <body>
        <h3 style="border: 3px solid; text-align: center;">List of workers</h3>
            <div class="mainBorders">
                <div>Name:ravi</div>
                <div>class:5th</div>
                <div>place:hyderabad</div>
                <div class="allImages worker1"></div>
                </div>
        </body>
</html>
```



➢ CSS Positions:

Till now in order to move the DOM Elements to a specified position we are using Padding and Margin properties. While using these Padding or Margin properties to move the DOM Element, it is not actually moving the DOM Element but it is increasing the dimensions.

In order to actually move the DOM Elements to a required position without increasing its dimensions we use the following CSS Properties **(Top, Left, Right and Bottom)**.

Not every DOM Element is capable of considering the above four CSS Properties but the DOM Element which are position can only consider above properties.

CSS **Position** is the property through which we could able to control the position of any DOM Element.

Following the possible value a position attribute takes,

1. Static
2. Relative
3. Absolute
4. Fixed
5. Sticky

➢ Element with position Static:

Every DOM Element by default holds the static position which indicates the DOM Element cannot be moved to any position from its default position. **(It will not consider the Top, Left, Right and Bottom properties).**

Syntax: **position : static;**

Ex:

➢ Element with position Relative:

Any DOM Element with position relative holds the following properties,

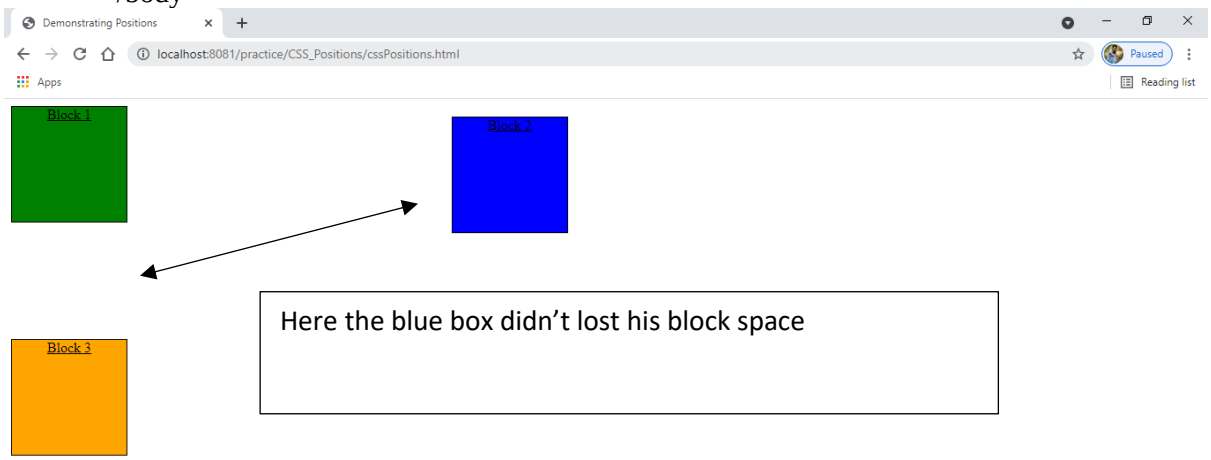
- It is capable of moving to any required position within the page. **(It considers the Top, Right, Left and Bottom properties).**
- While moving to a new position it never loses space been occupied on load of the page.
- While moving to a new position it always moves relevant to its default position.

Syntax: **position : relative;**

Ex:

```
<head>
    <title>
        Demonstrating Positions
    </title>
    <style type="text/css">
        .block {
            border: 1px solid;
            text-decoration: underline;
            text-align: center;
            width: 130px;
            height: 130px;
        }
        .block1 {
            background: green;
        }
        .block2 {
            background: blue;
            left: 500px;
            top: -120px;
            position: relative;
        }
        .block3 {
            background: orange;
        }
    </style>
</head>
<body>
    <div class="block block1">
        Block 1
```

```
        </div>
        <div class="block block2">
            Block 2
        </div>
        <div class="block block3">
            Block 3
        </div>
    </body>
```



➢ Element with position absolute:

Any DOM Element with position absolute holds the following properties,

- It is capable of moving to any required position.
- It considers Top, Left, Right and Bottom properties.
- Element with position absolute will automatically loses its default space it occupies in the page.
- While moving to a new position it always moves relevant to its parent position.
- While depending on the parent position it only depends on the parent whose position value is non-static.
- If its intermediate parent doesn't hold position on static, it travers to its grandparents until it finds an element or a parent with position non-static.

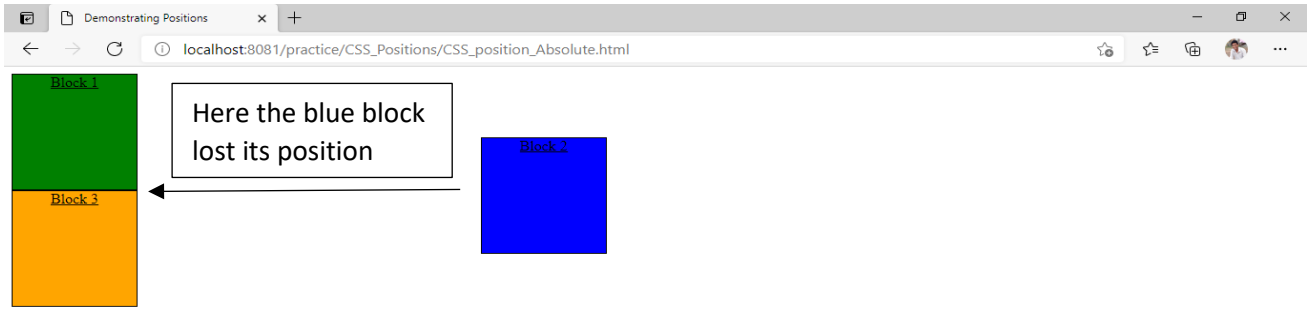
Note:

Elements with position absolute automatically jumps from default X-Y Axis to Z-Axis.

Syntax: **position : absolute;**

Ex:

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Demonstrating Positions
        </title>
        <style type="text/css">
            .block {
                border: 1px solid;
                text-decoration: underline;
                text-align: center;
                width: 130px;
                height: 130px;
            }
            .block1 {
                background: green;
            }
            .block2 {
                background: blue;
                left: 500px;
                top: 80px;
                position: absolute;
            }
            .block3 {
                background: orange;
            }
        </style>
    </head>
    <body>
        <div class="block block1">
            Block 1
        </div>
        <div class="block block2">
            Block 2
        </div>
        <div class="block block3">
            Block 3
        </div>
    </body>
</html>
```



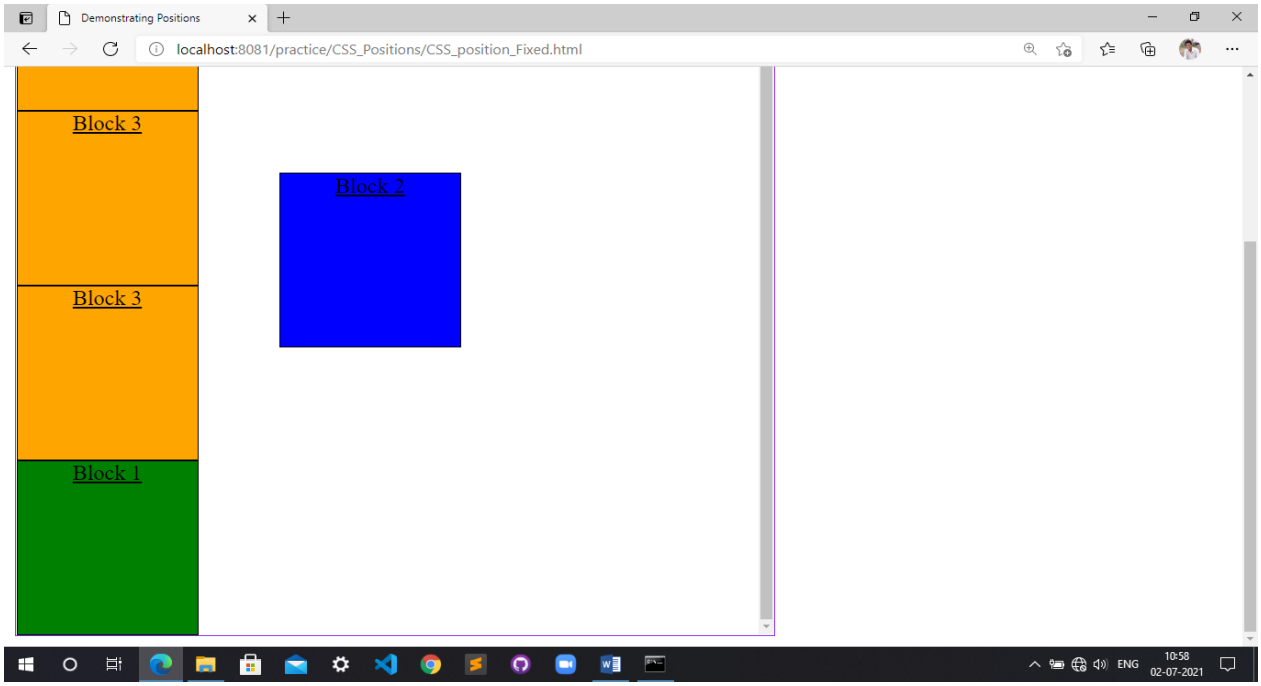
➢ Element with position fixed:

Any DOM Element with position fixed is almost like an element with position absolute. Where the only difference is once the element gets its fixed position it doesn't move from its original position even when we scroll.

Syntax: **position : fixed;**

Ex:

```
.block2 {
        background: blue;
        left: 200px;
        top: 80px;
        position: fixed;
}
```



➢ **Element with CSS Position Sticky:**

Any DOM Element with position sticky is almost like an element with position relative, where the only difference is once its position values been given (Top, Left, Right and bottom properties), if we try to scroll the element out of its view port, it automatically turns to fixed position and doesn't get scrolled.

Syntax: **position : sticky;**

Ex:

```
.block2 {
        background: blue;
        left: 200px;
        top: 80px;
        position: sticky;
}
```

➢ CSS Z-Index Property:

The elements which are falling under Z-Axis there is a chance of multiple elements override each other while rendering on the page.

While the Elements are overriding we can control the rendering order through CSS Z-Index Property.

Z-Index is a CSS Property takes a positive number as a value through which we can specify the priority order, the with the higher priority order always renders on the top.

Syntax: z-index: integer_value;

- **Note:**

Z-Index property can only be applied to elements which fall under Z-Axis (Elements with position Non-Static).

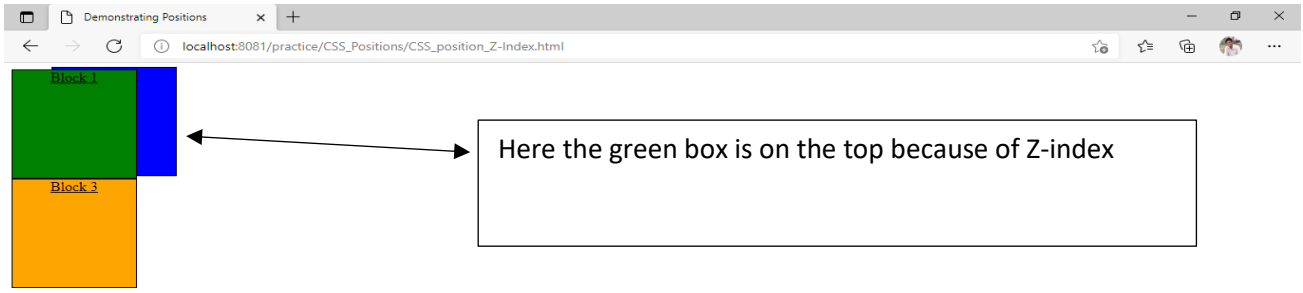
Ex: .block1 {

background: green;

position: relative;

z-index: 3;

}



➢ CSS Float Property:

By default while the elements getting rendered on the page they try to render from Left-Top Position of the container, the block level element fall under new line, Inline elements are gets rendered within the same line.

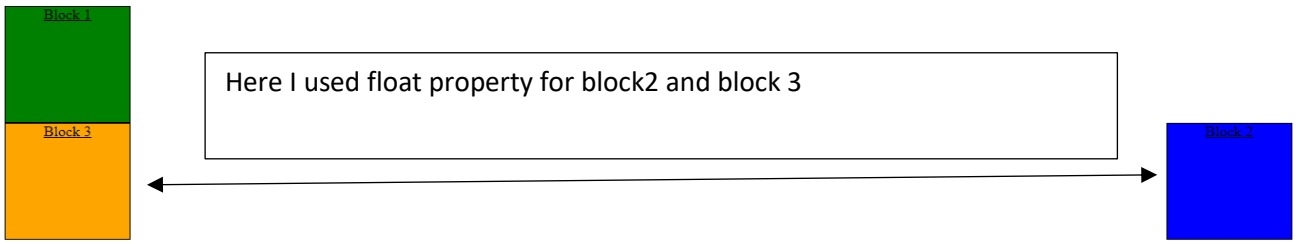
Using CSS Float Property we can make the elements to the right side of the container or to the left side of the container. Following are the possible values of a float property takes,

Syntax: float : left/right;

- Element with **float left** property will render to complete left side of the container, element with **float right** property will render to complete tight side of the container.
- Multiple continuous **DOM** Elements applied with **CSS Float** property will always to render with in the same line irrelevant of block or inline type of the Element.
- If an element changes its default direction through float property, the element which is following it will also try to render in the same line. To make the element not to follow previous element direction we make use of the CSS Property **clear**.
  Syntax: clear:left/right/both;

Ex:
```
.block1 {
        background: green;
        position: relative;
}
.block2 {
        background: blue;
        float: right;
}
.block3 {
        background: orange;
        float: left;
```

}



- **HTML Tables:**

Following are the predefined tags supported in HTML, using which we could able to render any data in the form of row wise and column wise.

- **<table>** Holds the complete table content.
- **<thead>** Holds the table header.
- **<tbody>** Holds the body of the table.
- **<th>** Holds the table header cell.
- **<tr>** Holds the row of the table.
- **<td>** Holds the table data cell.
- **<tfoot>** Holds footer of the table.

- **Structure of Basic table in HTML:**

```
<table>
    <thead>
        <tr>
            <th> head 1 </th>
            <th> head 2 </th>
                …
        </tr>
    </thead>
    <tbody>
        <tr>
            <td> data </td>
                …
        </tr>
        …
    </tbody>
    <tfoot>
```

```
            <tr>
                <td> … </td>
            </tr>
        </tfoot>
    </table>
```

Ex:

```
<body>
    <table border="2px solid">
        <thead>
            <tr>
                <th>Sno</th>
                <th>Date</th>
                <th>Time</th>
                <th>Present Location</th>
                <th>Past Location</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>1</td>
                <td rowspan="2">03-07-2021</td>
                <td>08:11 am</td>
                <td colspan="2">Vizag</td>
            </tr>
            <tr>
                <td>2</td>
                <td>09:11 am</td>
                <td>Hyderabad</td>
                <td>delhi</td>
            </tr>
        </tbody>
    </table>
```

- We can use border inside table attribute without style attribute.
- **Row Span and Col Span:**

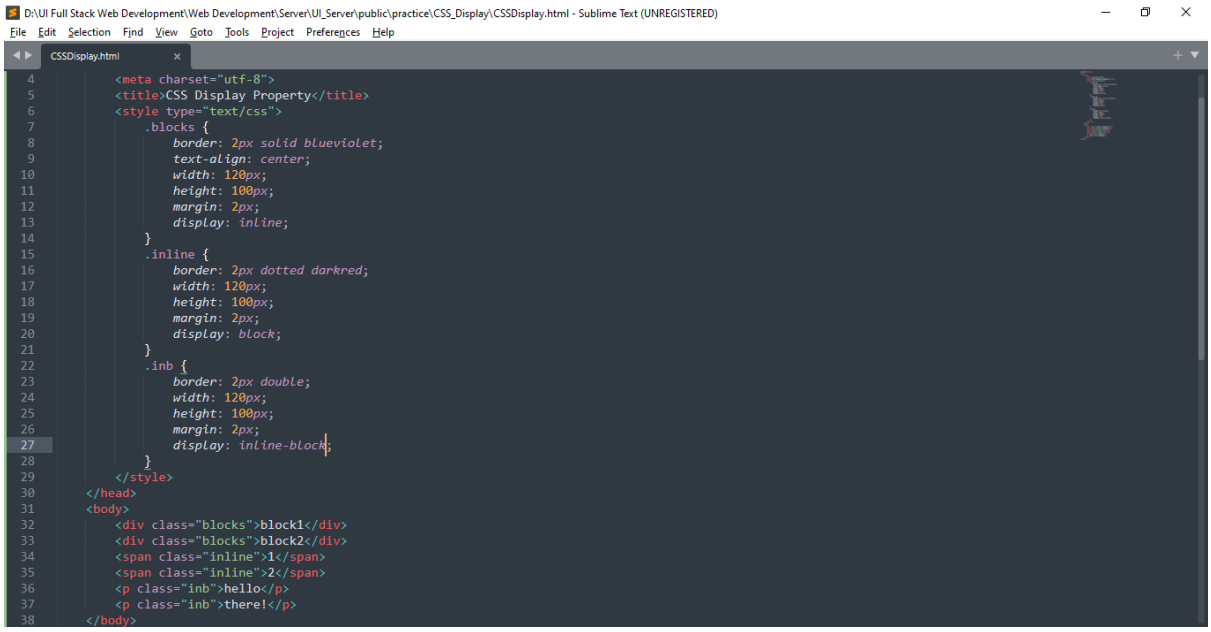
The rowspan and colspan are <td> tag attributes. These are used to specify the number of rows or columns a cell should span. The rowspan attribute is for rows as well as the colspan attribute is for columns.

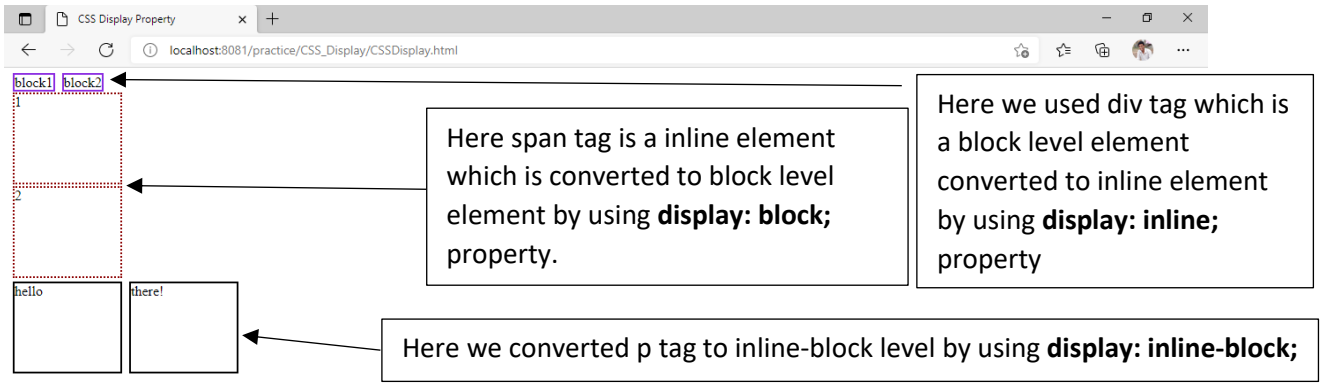
These attributes have numeric values, for example, colspan=3 will span three columns. As I mentioned in above Example.

➢ CSS Display Property:

'Display' is a CSS Property through which we could able to change the default rendering type of any DOM Element. Following that the possible values it takes,

- Display: Block;
  Makes the DOM Element to render like a block level element.
- Display: inline;
  Makes any DOM Element to render like an Inline Element.
- Display: inline-block;
  Makes any DOM Element to render like a Block Level Element, but occupies in the same line as like in Inline Element.



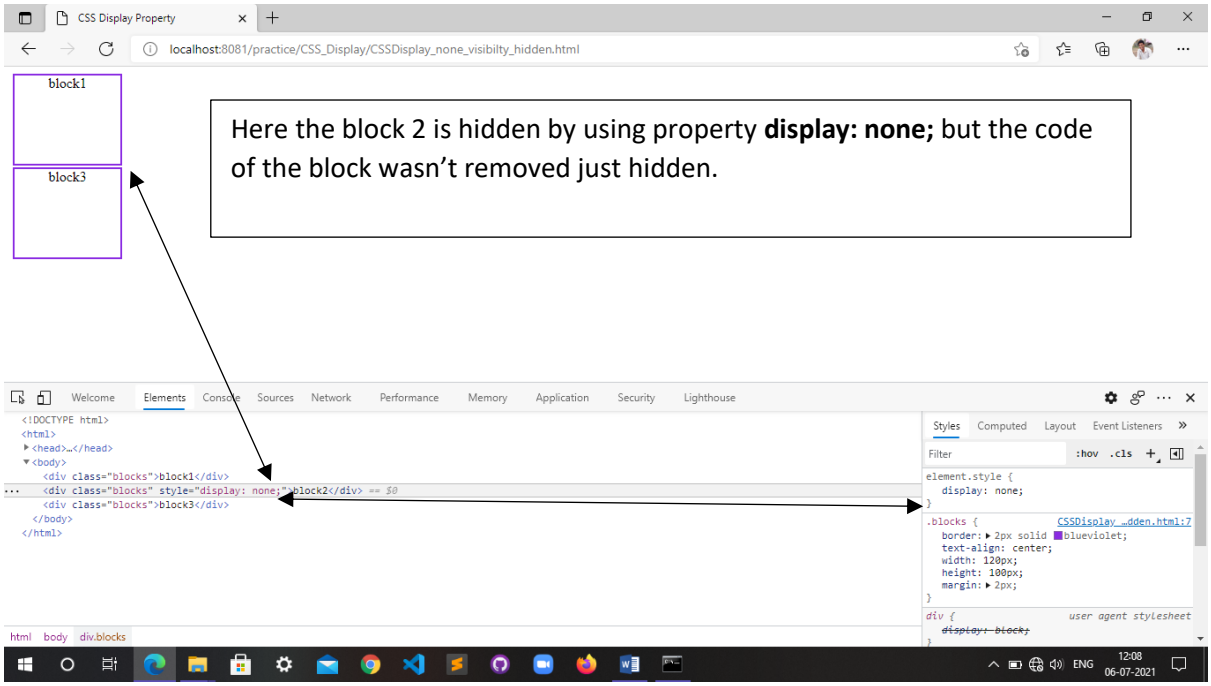


- Display: none;
  Makes element to not to be shown on the page. It still exist in DOM Structure).

- Display: Flex;

  A CSS3 Property to render flexible items on page.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>CSS Display Property</title>
        <style type="text/css">
            .blocks {
                border: 2px solid blueviolet;
                text-align: center;
                width: 120px;
                height: 100px;
                margin: 2px;
            }
        </style>
    </head>
    <body>
        <div class="blocks">block1</div>
        <div class="blocks" style="display: none;">block2</div>
        <div class="blocks">block3</div>
    </body>
</html>
```



➢ Difference between display: none and visibility hidden properties:

- Both CSS properties are used to make the DOM element to be not visible on the page where the only difference is display 'none' makes the element to be not visible on the page, it doesn't even occupy its own space.
- Visibility 'hidden' also makes the element to be not shown on the page but it still occupies its own space with in the page.

Note:

Both visibility hidden and display 'none' properties makes element to be not visible on the page but the element still exists within the DOM structure.

➢ CSS Pseudo Classes:

Following are the predefined pseudo classes been supported using which we could able to apply the CSS on elements not on load of the page but based on current state of the element.

- **:hover :**
  Applies set of CSS when there is a hover
- **:empty :**
  Applied to any element which doesn't have child element
- **:disabled :**
  Applied to element with disabled state
- **:enabled :**
  Applied to element with enabled state
- **:active :**
  Selects only active link
- **:checked :**
  Selects checkbox element with checked state
- **:focus :**
  Selects the element where it is in focus state
- **:first-child :**
  Element which is in first child state
- **:last-child :**
  Element which is in last child state
- **Div:first-of-type :**
  To select every div element which is the first div of its parent
- **a:link :**
  Selects unvisited links
- **a:visited :**
  Selects visited links
- **n^{th} – child(2) :**
  Selects element in second position.
- **P:only-child :**
  Selects p tag which is only child of its element.

➢ Pseudo Elements:

Following are the predefined Pseudo Elements been supported using which we could able to apply CSS for not to the Complete Element but partially to the content of the element.

- **::after :**
  Add any element after the selected element.
- **::before :**
  Add any element before the selected element.
- **::first-letter :**
  Only applies to the first letter of the container.
- **::first-line :**
  Only applies to the first line of the content.
- **::marker :**
  Only applies to set CSS for markers of the list items.
- **::selection :**
  Select the part/portion of the element content which is selected.

```
        <title>Pseudo Elements</title>
        <style type="text/css">
            p::first-letter {
                font-size: 50px;
            }
            p::first-line {
                color: red;
            }
            ::marker {
                color: green;
                font-weight: bolder;
            }
            p::after {
                content: 'and how are you?';
                color: black;
            }
            p::before {
                content: 'Hello! ';
                color: violet;
            }
            p::selection {
                background: ghostwhite;
                color: darkviolet;
                font-weight: bolder;
            }
        </style>
    </head>
    <body>
        <p>this is Ananth </p>
        <ol>
            <li>list item1</li>
            <li>list item2</li>
            <li>list item3</li>
        </ol>
    </body>
```

Hello! this is Ananth and how are you?

1. list item1
2. list item2
3. list item3

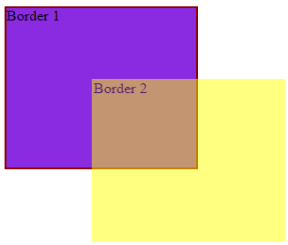
➢ CSS Opacity Property:

While elements get override each other we could able to control the transparency level of the elements through CSS Opacity property.

It takes a value between 0 to 1

Ex: opacity: 0.2;

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>CSS_OPACITY</title>
        <style type="text/css">
            .Border1 {
                border: 2px solid darkred;
                background-color: blueviolet;
                height: 180px;
                width: 200px;
            }
            .Border2 {
                border: 2px solid yellow;
                background-color: yellow;
                height: 180px;
                width: 200px;
                position: absolute;
                left: 100px;
                top: 90px;
                opacity: 0.5;
            }
        </style>
    </head>
    <body>
        <div class="Border1">Border 1</div>
        <div class="Border2">Border 2</div>
    </body>
</html>
```



➢ HTML Input Elements:

Following are the Pre-Defined HTML Elements supported using which we could able to read different types of data from the user.

- **<input type=”text” >**
  To read text type of content.
- **<input type=”password”>**
  To read sensitive data.
- **<input type=”date”>**
- **<input type=”radio”>**
  Creates radio button.
- **<input type=”checkbox”>**
  Creates check box.

- **<input type=”button”>**
  Creates a button.
- **<input type=”submit”>**
- **textarea**
  Creates a multiline text container.
- **<select>**
  <option>option1</option>
  <option>option2</option>

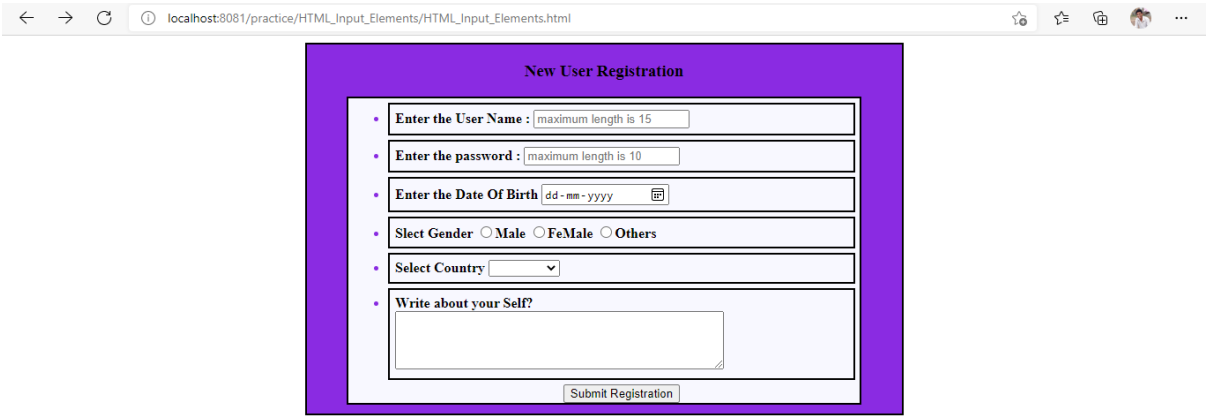
  </select>

  Creates a dropdown with multiple options.

Ex: <form action="http://www.serverlocation.in.com" method="Post">

```
<ul>
<li>
<label for="username"><b>Enter the User Name :</b></label>
<input type="text" name="username" placeholder="maximum
length is 15" maxlength="15">
</li>
<li>
<label for="psw"><b>Enter the password :</b></label>
<input type="password" name="psw" placeholder="maximum
length is 10" maxlength="10" title="password should contain
1upper case and lower case and special symbol and number">
</li>
<li>
<label for="date"><b>Enter the Date Of Birth</b></label>
<input type="date" name="date">
</li>
<li>
<label for="gender">Slect Gender</label>
<input type="radio" name="gender">Male
<input type="radio" name="gender">FeMale
<input type="radio" name="gender">Others
</li>
<li>
<label>Select Country</label>
<select title="click to get country name">
<option></option>
<option>Australia</option>
<option>China</option>
<option>India</option>
```

```
<option>Sri Lanka</option>
<option>England</option>
<option>Pakistan</option>
<option>UAE</option>
<option>Canada</option>
</select>
</li>
<li>
<label for="text">Write about your Self?</label><br>
<textarea name="text" rows="4" cols="50"></textarea>
</li>
<div style="text-align: center;"><input type="submit"
name="submit" value="Submit Registration"></div>
</ul>
</form>
```



- **HTML Form Tag:**

A predefined tag using which we could able to send user input data to the server.

It takes the following mandatories i.e. method and action.

METHOD attribute used to which we could specify the type of the communication while sending or receiving the data.

ACTION attribute used for through which we specify the path of the server to which communication should happen.

Ex: <form action="server url" method="get/post">

……

……

</form>

- **Types of Communication:**

While communicating with a server, it could be either secured or non-secured type of communication.

- **Non-Secured type of Communication (GET):**

  In this type of communication the data will be sent to the server by appending to the URL as query parameters.
  Ex: http://www.abc.com/data/user/info?uname=test&age=20....
- **Secured Communication(POST):**

  In this type of communication the data will be sent to the server by adding it to the request header which is not exposed to the end user.

  Any time we send sensitive data to the server we use **POST** type of communication.

Java Scripts

➢ What is Java Script?

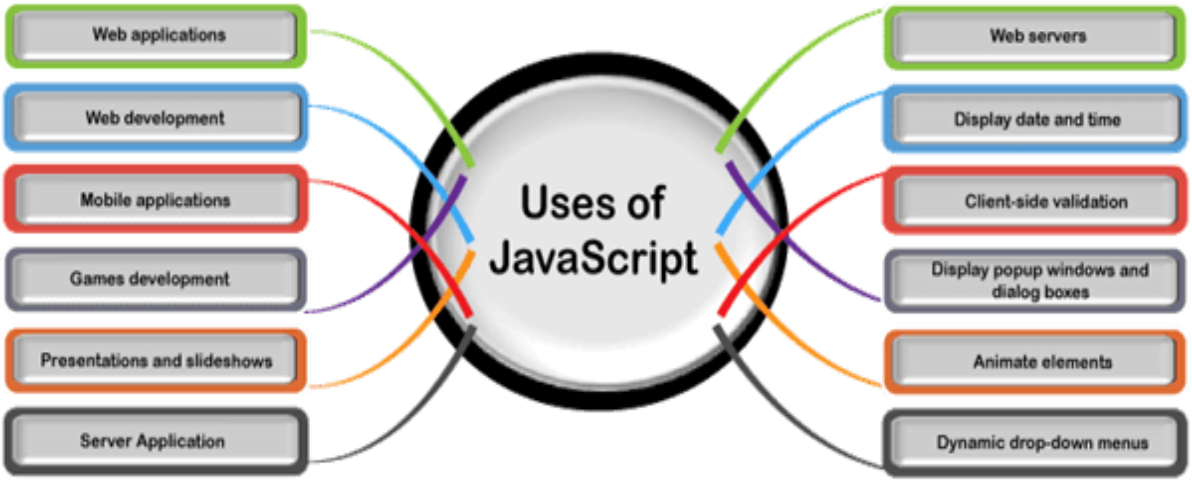
A predefined scripting language where we can implement logical ability, arithmetic operations, DOM Operations in a Web Page.

- It is a client side scripting language capable of executing with in the client side itself.
- It is the only programming language can be understand by the browser.
- By default every browser comes with JavaScript Engine through which JavaScript instructions gets compiled and executed within client side.
- It comes with predefined objects and methods through which DOM Operations can be performed on the page.
- It is super heroic Programming Language capable of creating, updating or deleting any HTML Element or its corresponding CSS Properties.
- It comes with predefined objects and methods through which we could implement AJAX calls within the Page.
- JavaScript's follows top to bottom execution process, it doesn't hold the concept of having predefined static points (like init or main method).

➢ What are the uses of JavaScript?

- JavaScript is a light-weight object-oriented programming language that is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language. JavaScript enables dynamic interactivity on websites when it is applied to an HTML document.
- JavaScript helps the users to build modern web applications to interact directly without reloading the page every time. JavaScript is commonly used to dynamically modify HTML and CSS to update a user interface by the DOM API. It is mainly used in web applications.

Let's discuss the uses of JavaScript. Some of the uses of JavaScript are representing in the following image.



- **Web Applications:**

As day-by-day there is a continuous improvement in the browsers, so JavaScript gained popularity for making robust web applications. We can understand it by taking the example of **Google Maps**. In Maps user just requires to click and drag the mouse; the details are visible just by a click. There is a use of JavaScript behind these concepts.

- **Web Development:**

JavaScript is commonly used for creating web pages. It allows us to add dynamic behaviour to the webpage and add special effects to the webpage. On websites, it is mainly used for validation purposes. JavaScript helps us to execute complex actions and also enables the interaction of websites with visitors. Using JavaScript, it is also possible to load the content in a document without reloading the webpage.

- **Mobile Applications:**

Now a day's mobile devices are broadly used for accessing the internet. Using JavaScript, we can also build an application for non-web contexts. The features and uses of JavaScript make it a powerful tool for creating mobile applications. The **React Native** is the widely used JavaScript framework for creating mobile applications. Using **React Native,** we can build mobile applications for different operating systems. We do not require writing different codes for the iOS and Android operating systems. We only need to write it once and run it on different platforms.

- **Server Applications:**

A large number of web applications have a server-side to them. JavaScript is used to generate content and handle HTTP requests. JavaScript can also run on servers through **Node.js**. The **Node.js** provides an environment containing the necessary tools required for JavaScript to run on servers.

- **Web Servers:**

A web server can be created by using **Node.js**. Node.js is event-driven and not waits for the response of the previous call. The servers created using Node.js are fast and don't use buffering and transfer chunks of data. The HTTP module can be used to create the server by using the **createServer()** method. This method executes when someone tries to access the port 8080. As a response, the HTTP server should display HTML and should be included in the HTTP header.

In this article, we discussed various JavaScript applications. JavaScript has various other uses that help us to improve the performance of webpages. The other uses of JavaScript are listed as follows:

- Client-side validation.
- Displaying date and time.
- To validate the user input before submission of the form.
- Open and close new windows.
- To display dialog boxes and pop-up windows.
- To change the appearance of HTML documents.
- To create the forms that respond to user input without accessing the server.

- **How to inject JavaScript in HTML:**

- Using inline JavaScript under the **<script>** tag,

Syntax: <script type = "text/javascript">

```
//JavaScript code

</script>
```

- In a single page any number of script tags can be placed.
- Inside the script tag we can use 'n' number of lines of JavaScript code.

- Script tag can be placed anywhere within the page.

➢ Data types in JavaScript:

Data types specifies the type of data that a variable can hold. In JavaScript while declaring a variable it is not required to specify the data type while declaring the variable.

'var' is a predefined keyword through which we could declare variable in JavaScript.

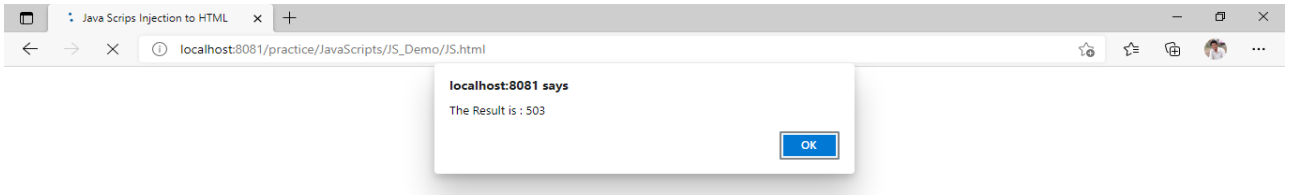
Following are the differ types of data types supported by JavaScript,

1. Number 2. String 3. Object 4. Undefined

5. Null 6. Boolean 7. Function.

Ex:

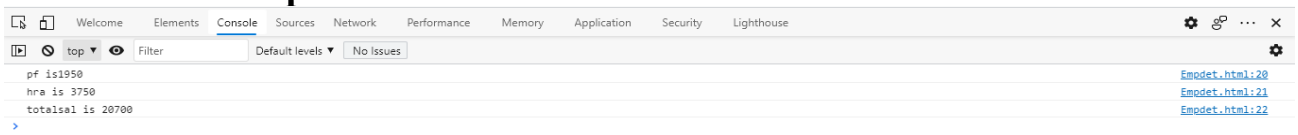
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Java Scrips Injection to HTML</title>
    </head>
    <body>
        <h3>java script inject</h3>
        <script type="text/javascript">
            var firstnum = 10;
            var secondnum = 493;
            var sum = firstnum + secondnum;
            alert("The Result is : " + sum);
        </script>
    </body>
</html>
```



➢ Write a program for employee details like ename, eage, basicsal, totalsal, hra, pf and department calculate total salary?

```
<script type="text/javascript">
        var ename ="ananth";
        var eage = 23;
        var gender = "Male";
        var dept = 'IT';
        var basicsal = 15000;

        var totalsal, hra, pf;
        pf = 13/100 * basicsal;
        hra = 25/100 * basicsal;
        totalsal = hra + basicsal + pf;
```

```
        console.log("pf is" + pf);
        console.log("hra is " + hra);
        console.log("totalsal is " + totalsal);
    </script>
```



➢ Type of Method:

A predefined method being supported in JavaScript using which we could able to get type of data a variable is holding.

It takes a variable as a parameter and returns, the type of data the variable is holding.

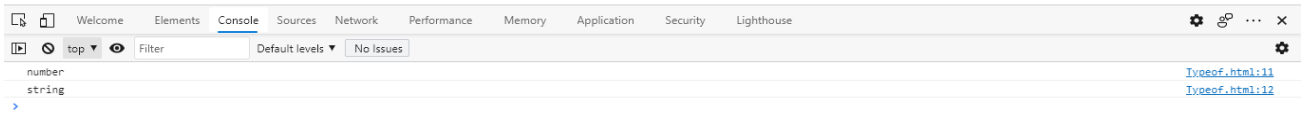
Syntax: typeof(Variable_Name);

Ex: var num = 10;

var name = ‘ananth’;

console.log**(typeof(age))**;

console.log**(typeof(name))**;

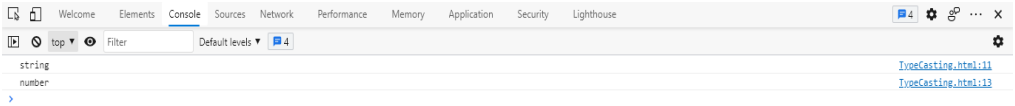


- If any variable is not initialized it will be defined as undefined.

➢ Type Casting:

Converting a data type into another is known as type casting. Sometimes there is a need to convert the data type of one value to another. Under some circumstances JavaScript will perform automatic type conversion.

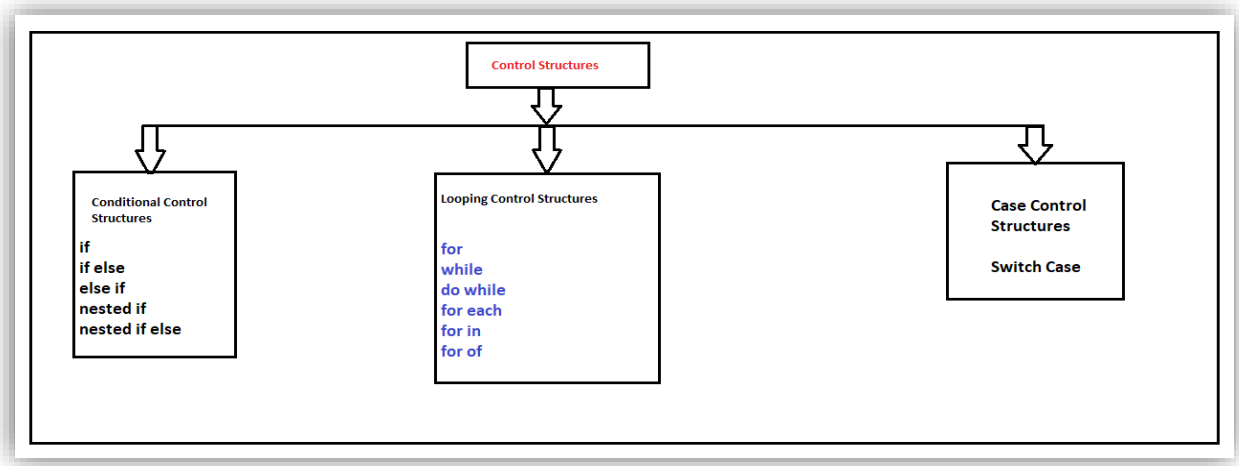
```
Ex: <script type="text/javascript">
        var num = "10";
        var name = 'ananth';
        console.log(typeof(num));
        num = parseInt(num);
        console.log(typeof(num));
    </script>
```



➢ Control Structures:

Following are the set of Control Structures being supported through which we could able to control the sequence of execution flow in a application.

1. Conditional Control Structures.
2. Looping Control Structures.
3. Case Control Structures.



➢ IF Condition:

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax: if (condition) {

// block of code to be executed if the condition is true

}

➢ Else :

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax: if (condition) {

// block of code to be executed if the condition is true

} else {

// block of code to be executed if the condition is false

}

- **Else if Statement :**

Use the else if statement to specify a new condition if the first condition is false.

Syntax: if (condition1) {

// block of code to be executed if the condition1 is true.

} else if (condition2) {

// block of code to be executed if the condition1 is false and condition 2 is true.

} else {

// block of code to be executed if the condition1 is false and condition 2 is False.

}

- **Nested if:**

A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, JavaScript allows us to nest if statements within if statements. i.e., we can place an if statement inside another if statement.

Syntax: if (condition1) {

// block of code to be executed if the condition1 is true.

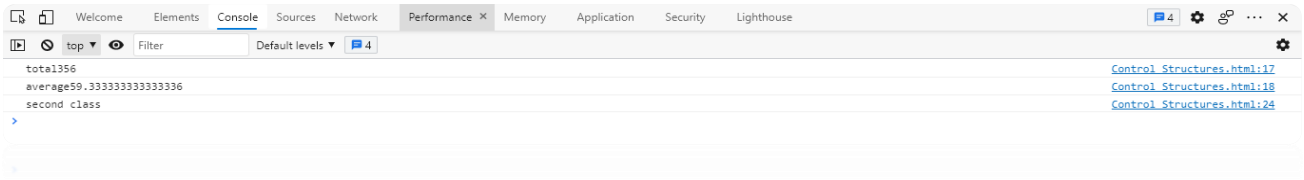
} if (condition2) {

// block of code to be executed if the condition 2 is true.

}

Ex:

```
<script type="text/javascript">
    var sname;
    var sage;
    var tel=90, hin=80, eng=51, maths=55, science=40, social=40;
    var avg;
    var total;
    total = tel + hin + eng + maths + science + social;
    avg = (total / 600)*100;
    console.log("total" + total);
    console.log("average" + avg);
    if (avg>70) {
        console.log("passed in distinction");
    } if (avg>60 && avg<70) {
        console.log ("passed in first class");
    } else {
        console.log ("second class");
    }
</script>
```



➢ Looping Control Structures:

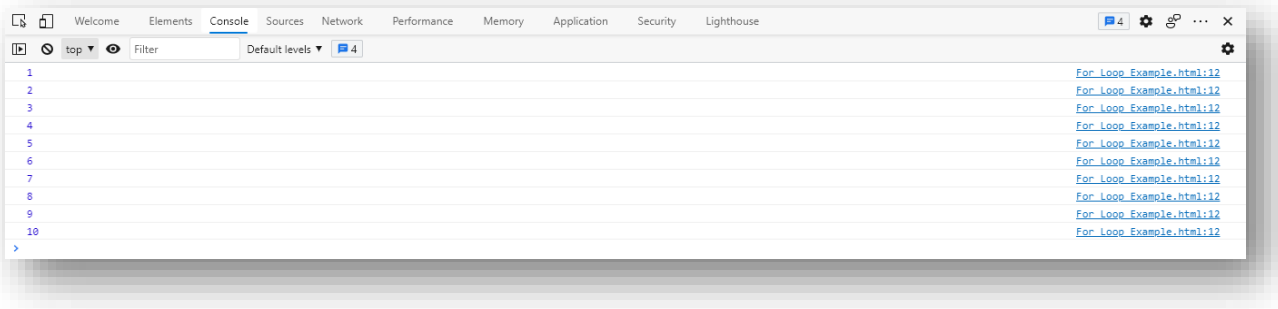
Looping Control Structures been supported in Java Script takes the set of instructions and repeat them for multiple times.

- **For Loop:**

Loops through a block of code a number of times.

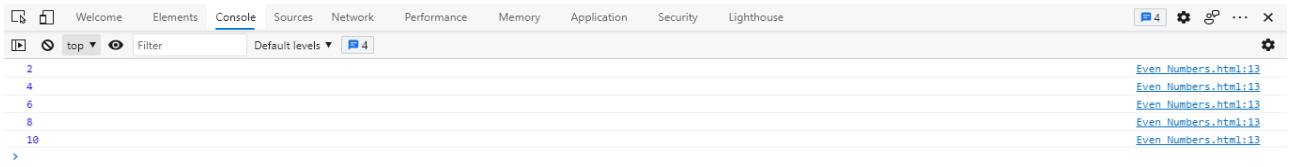
Syntax: for (initialization, <condition>, increment/decrement) {
// set of lines to be Executed
}

Ex:
```
<script type="text/javascript">
    var i;
    var j=10;
    for (i=1; i<=j; i++){
        console.log(i);
    }
</script>
```



➢ Write a program to print even numbers from 1 to 10?

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Even Numbers</title>
    </head>
    <body>
        <script type="text/javascript">
            var i;
            var j=10;
            for (i=1; i<=j; i++){
                if (i % 2 == 0 ) {
                    console.log(i);
                }
            }
        </script>
    </body>
</html>
```



- **Difference between "=", "= =" and "= = =":**
  - '=' it is an assignment operator using which we could able to assign a value of right side to the variable of left side.
    Ex: var a=10;
  - '==' it is an comparison operator used to compare value of right side and returns a Boolean value (true/false) based on the equality of the values.
    Ex: var a = 10;
    Var b = 11;
    a==b; //returns false
    Note: '==' operator, while comparing values it only checks for value equality but not data type equality.
  - '===' it is almost like an equality '==' operator used to compare value equality were the only difference is while comparing values it will not just checks for value equality but it also checks for data type equality.
    Ex: var a = "10";
    var b = 10;
    a===b; // returns false.

- **While Loop Control Structure:**

It is almost like a for loop control structure issued to iterate instructions more than once were the only difference is while loop takes only the condition were the increment and decrement can be placed anywhere within the code.

Syntax: while (condition) {

- - - - -

}

Ex: WAP to print the lucky number of a given number?

```
<script type="text/javascript">
    var number = 12345;
    var sum = 0, a;
    while (number > 0) {
        a = number % 10;
        sum += a;
        number = parseInt(number/10);

        if (number == 0 && sum >= 10) {
            number = sum;
            sum = 0;
        }
    }
    console.log(sum);
```

- **WAP to find the reverse of a number?**
- **WAP to check weather given number is palindrome or not?**
- **WAP to list out all palindrome numbers from 300 to 21?**
- **WAP to list out reverse numbers between 32 to 950?**
- **WAP to check Weather the given number is Armstrong or not?**

- **Reading values through prompt:**

The prompt () method displays a dialog box that prompts the visitor for input.

A prompt box is often used if you want the user to input a value before entering a page.

By default it takes user input as string.

Note: When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. Do not overuse this method, as it prevents the user from accessing other parts of the page until the box is closed.

Syntax: var variable_Name = prompt ("text");

- **Do-while:**

It is almost like a while loop control structure used to repeat a set of instructions for more than once were the only difference is while loop will execute the instructions only when the provided condition is true, were as do-while executes the set of instructions at least for once irrelevant of whether provided condition is true or false.

Syntax: do {

//set of instructions is executed for atleast once irrelevant to the provided condition is true or false

} While (condition);

- **Case Control Structure:**

It takes multiple cases within it and invokes corresponding/particular case based on the value been passed.

Syntax: switch (<value>) {

Case <value> :

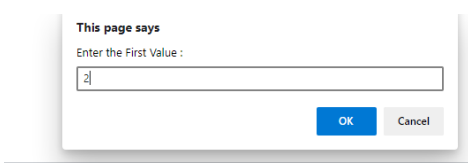
//stmts

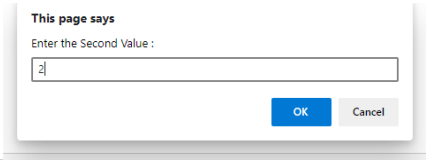
//stmts

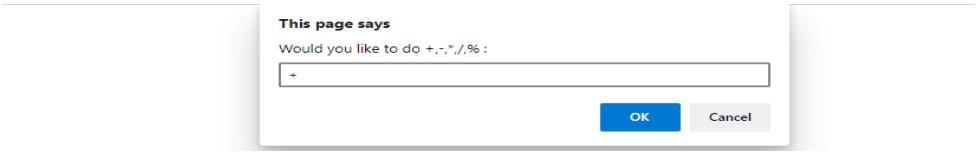
Break;

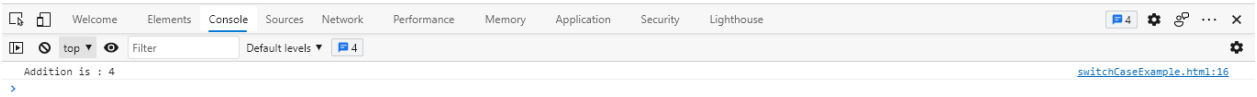
}

```
<script type="text/javascript">
    var firstValue = parseInt(prompt("Enter the First Value : "));
    var secValue = parseInt(prompt("Enter the Second Value : "));
    var result;
    var uinput = prompt ("Would you like to do +,-,*,/,% : ");
    switch (uinput) {
        case '+' :
            result = firstValue + secValue;
            console.log("Addition is : " + result);
            break;
        case '-' :
            result = firstValue + secValue;
            console.log("Substraction is : " - result);
            break;
        case '*' :
            result = firstValue * secValue;
            console.log("Multiplication is : " + result);
            break;
        case '/' :
            result = firstValue / secValue;
            console.log("Division is : " + result);
            break;
        case '%' :
            result = firstValue % secValue;
            console.log("Modulous is : " + result);
            break;
        default :
            console.log("Wrong option selected..... Please try again.")
    }
</script>
```









➢ Variable Hosting:

Most of the programming languages forces to declare a variable only at the starting of the program so that it can allocate the memory and then works on variable.

- In JavaScript, it is not mandatory to declare the variables only at the starting of the program but can be declared any were we wanted.
- While executing the application JavaScript performs variable hosting process in which it identifies all the variable declarations with in the application and moves all the declarations to starting of the applications.
- The process of moving all the variable declarations to the starting of the application is called variable hosting.

Note:

While moving the declarations it only moves declarations to the starting but not the initialized values.

➢ Data Structures:

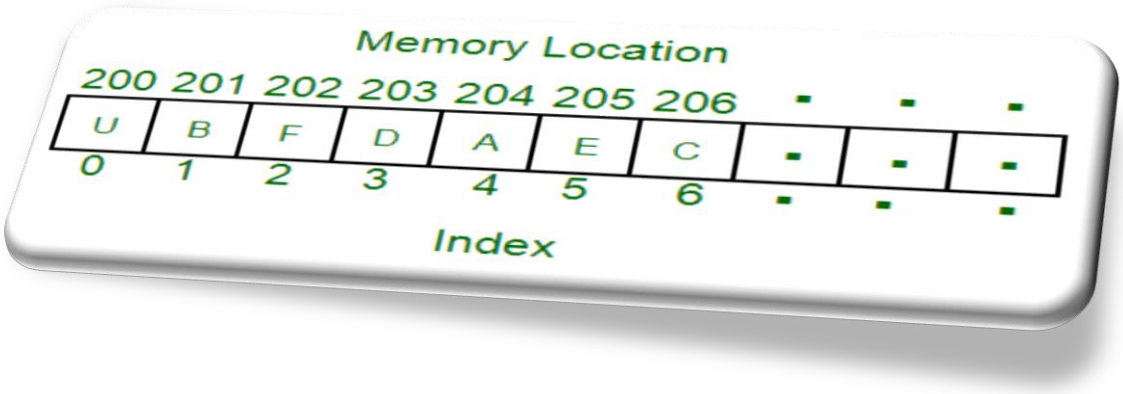
Following are the different data structures being supported in the JavaScript through which different type of data can be stored and retrieved.

- Arrays
- Json
- Map
- Set
- Tables

➢ Array Data Structure:

A predefined data structure through which we could able to hold group of multiple values within it.

- In JavaScript, arrays are the **heterogeneous data structures** capable of holding different types of data with in it.
- Array is capable of holding different and multiple types of data, every other value will be identified with single variable name.
- All the values within the array will be automatically assigned with a numeric **index** value.
- The **index** value is always starts with "Zero".
- '[ ]' are always used to represent the arrays in any language.
- Within the memory, arrays always occupy continuous memory allocation for all the values within it.
- We make use combination of array name and corresponding index position while working with an array.



- Following are the two different ways we can create an array in JavaScript,

  var a = new array( ) **//dynamic allocation**

  (OR)

  var a = [ ]; // **empty array**

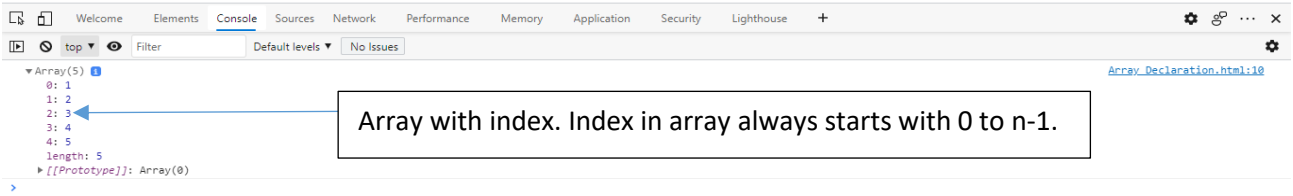
  var a = [1, 2, 3, 4]; **//declaring an array with values.**

Ex: **<script type="text/javascript">**

var a = [1, 2, 3, 4, 5];

console.log(a);

</script>



➢ Pre-Defined methods supports in array:

Following are the some predefined methods supported by array in JavaScript,

- **Length:**

  It is a property which returns the total number of values in an array.

  Syntax: arrayName.**length;**

- **Push(<values>):**

  Used to insert single or multiple values to an array from right direction.

  Syntax: arrayName.push(<value>);

- **Pop():**

  Used to delete single value from an array from right side.

  Syntax: arrayName.pop();

- **Shift():**

  Used to delete single value from an array from left direction.

  Syntax: arrayName.shift();

- **Unshift(<values>):**

  Used to insert single/multiple values to an array from left direction.

  Syntax: arrayName.unshift(<values>);

- **Splice(starting position, <no.of.values to be deleted>,<optional values to be inserted>):**

  Used to insert or delete values.

  Syntax: arrayName.splice(starting position, <no.of.values to be deleted>,<optional values to be inserted>);

- **Join():**

  Used to merge all the values of array to a single value with/without separator.

➢ **<u>Predefined methods can be applied on string:</u>**

Every string being created in JavaScript internally it gets created in the form of an array. Following are the predefined methods can be applied on string in JavaScript,

- **Length:**

  It returns the total number of characters within a string, including spaces.

- **charAt(<index>):**

  Returns the character present at the provided index position.

- **charCodeAt(<index>):**

  Returns the ASCII value of a character present at provided index position.

- **subStr(index,length of substring):**

  Returns the substring from the given string.

- **replace("what","with"):**

  Used to replace the string parts with required strings.

- **Split("optional separator"):**

  Used to split the main string in the form of an array.

- **Includes("<string>"):**

  Checks weather the provided string is part of main string and returns true or false.

- **Indexof("string"):**

  Returns at what index provided substring exist in main string.

- **tolowercase():**

  Converts the provided string to lower case format.

- **touppercase():**

  Converts the provided string to upper case format.

- **Slice(start index, end index):**

  Returns the substring from one index to other.

✓ **Note:**

Any string being defined in JavaScript it occupies the memory internally as like an array where each character occupies individual block with the corresponding index value.

➢ **Working with JavaScript "Date" object:**

"Date" is a predefined class supported in JavaScript using which we could able to work with the systems current date or user defined custom date.

Syntax:

var date = new Date(); //creates a reference of systems current date.

var date = new Date(<user specific date>); // user specific date

Following are the predefined methods which can be applied on date object,

- **get Date():**

  Returns the current date value(1^{st} -31^{st}).

- **get Month():**

  Returns the current month value(0^{th}-11^{th}).

- **getFullYear():**

  Returns the full year

- **get Hours():**

  Returns the hours value.

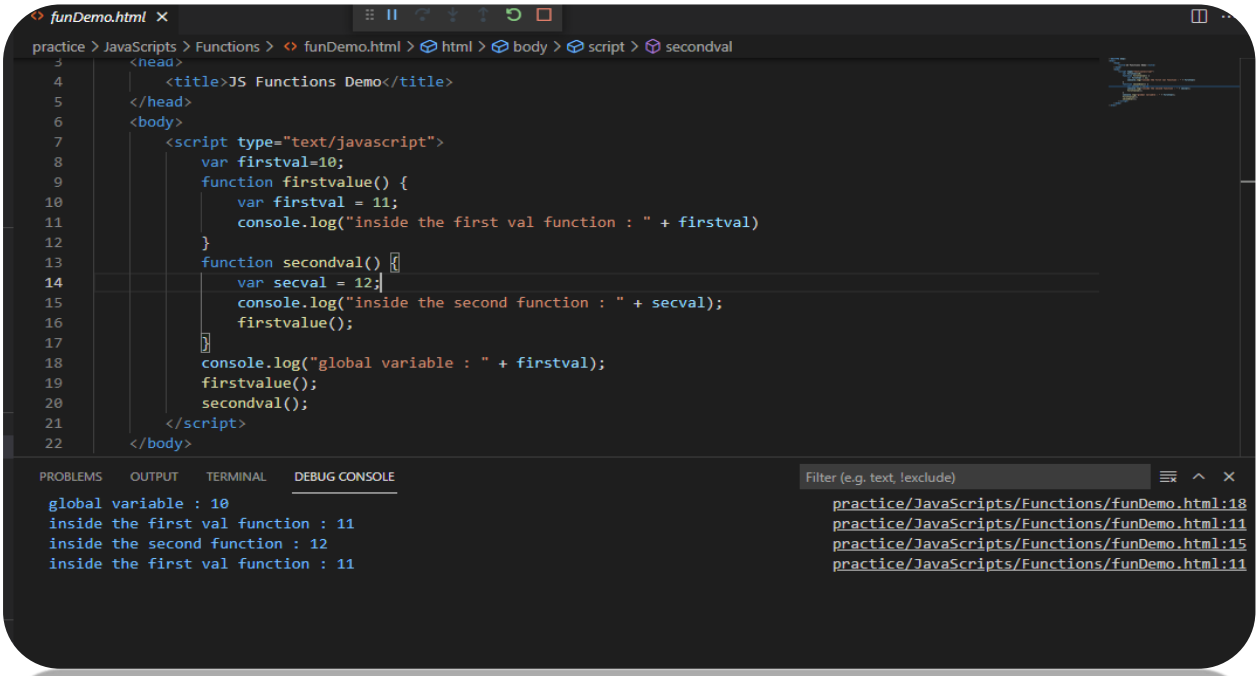
- **get Minutes():** Returns 0-59
- **get Seconds():** Returns 0-59
- **get Milliseconds():** Returns 0-999
- **get Day():** Returns 0-6.

➢ **JavaScript functions:**

The set of instructions binded as a module with in "{ }" assigned with an user defined custom name which can be reused anywhere within the page is called a function.

- In JavaScript **'function'** is a predefined keyword using which we can define a function.**[until ECMA6].**
- In a single page we can define any number of functions in JavaScript.
- Once the function ben defined it can be invoked any number of times through the function name.
- While defining the function we name it through **function** keyword, while calling we simply use function name.

Ex:



➢ Local variables and global variables:

Any variable which is declared outside of a module will become global variable can be accessible anywhere within the application.

Global variable can be accessible and modifiable anywhere within the page.

Local variable can't be accessed outside the module it can be accessed within the module.

Ex: var a = 100 //global variable

Function test(){

var a = 101; //local variable

}

➢ Accessing private data of a function outside of it:

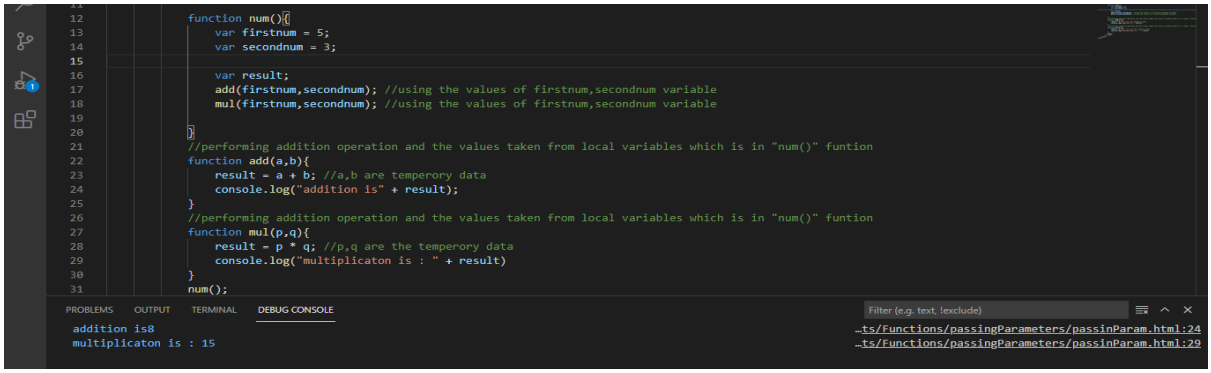
- Through passing parameters while calling a function.
- Returning value from called function to calling function.
- Both passing parameters and returning values.

➢ Passing parameters while calling a function:

While calling a function within another function we could able to pass local-variables of calling function to called function.

- While calling a function we can pass any number of parameters.
- While defining a function we could able to catch the values being passed while calling a function, using temporary variables.
- The parameters being passed while calling a function are called as actual parameters whereas the temporary parameter declared at function definition are called formal parameters.

Ex:



- While passing formal parameters at function definition we don't need to specify or declare those parameters we simply specify the variable names.

➢ **Returning a value from the called function to calling function:**

```
<script type = "text/JavaScript">
    function num() {
        var first = 2;
        var second = 4;
        var temp = add(first,second);
        console.log("addition is " + temp);
        var temp1 = mul(first,second);
        console.log("multiplication is " + temp1);

    }
    function add(a,b){
        var result;
        var result = a + b;
        return result;
    }
    function mul(p,q){
        var result;
        result = p * q;
        return result;
    }
    num();
</script>
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE        Filter (e.g. text, !exclude)   Open UsingReturn.html
addition is 6                                   ...ts/Functions/passingParameters/UsingReturn.html:11
multiplication is 8                             ...ts/Functions/passingParameters/UsingReturn.html:13
```

➢ **"Arguments" Keyword:**

A predefined keyword supported in JavaScript and by default available in every function which holds all the parameter values been passed to that function in the form of an array.

- In JavaScript it is mandatory to catch the parameters in function definition while matching function call with function definition it only matches function name but never looks for type of parameter or number of parameters being passed.
- Irrelevant of whether the parameters been catched or not, arguments keyword gets created in every function and holds all the parameters being passed to it in the form of an array.

```
gumentKeyword.html
cripts > Functions > <> argumentKeyword.html > html > body > script > addValues
<!DOCTYPE html>
<html>
  <head>
    <title>JS Functions Demo</title>
  </head>
  <body>
    <script type="text/javascript">
      function test() {
        console.log(arguments);
      }
      function addValues() {
        var result = 0;
        for (var i = 0; i < arguments.length; i++) {
          result += arguments[i];
        }
        // result = a + b + c;
        console.log("The sum is " + result);
      }

      addValues(34, 78, 23, 33, 55, 12);
      test(23, 44);
    </script>
```

➢ JSON Data Structure:

JSON is derived as **JavaScript Object Notation.**

- It is **predefined data structure** being supported in JavaScript by default.
- It is used to not just hold the data but, holds the data with the extra information of particular data.
- The JSON Data gets stored in the format of **key and value** based.
  Ex: key: value,
  Key2: value2;
- All the keys within the JSON Object should be only **string** data type, whereas the **corresponding value** could be of any **JavaScript supported Data Type**.
- While accessing data from object we make use of the combination of object name and corresponding key.

Ex:

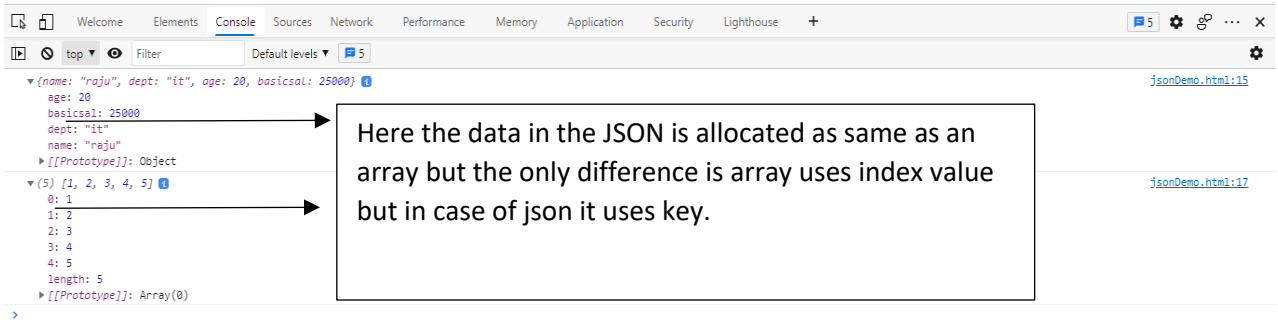
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <script type="text/javascript">
            var empdetails = {
                'name' : 'raju',
                'dept' : 'it',
                'age' : 20,
                'basicsal' : 25000,
            }
            empdetails.hra = 15/100 * empdetails.basicsal;
            empdetails.pf = 25/100 * empdetails.basicsal;
            empdetails.total = empdetails.basicsal + empdetails.hra + empdetails.pf;
            if (empdetails.total > 20000) {
                empdetails.tax = empdetails.total / 15;
            } else {
                empdetails.tax = 0;
            }
            console.log("name" + empdetails.name);
            console.log("dept" + empdetails.dept);
            console.log("age" + empdetails.age);
            console.log("basic salary" + empdetails.basicsal);
            console.log("hra is " + empdetails.hra);
            console.log("pf is" + empdetails.pf);
            console.log("total is" + empdetails.total);
            console.log("tax is :" + empdetails.tax);
        </script>
    </body>
</html>
```

➢ Memory allocation of JSON object:

Internally JSON object gets memory allocated same as like an array where the only difference is, the key of the data is assigned as index value by default.

Ex:

```
        <script type="text/javascript">
            var empdetails = {
                'name' : 'raju',
                'dept' : 'it',
                'age' : 20,
                'basicsal' : 25000,
            }
            console.log(empdetails)
            var a = [1, 2, 3, 4, 5];
            console.log(a);
        </script>
```



➢ Anonymous function:

Any function which is defined without a name is called an anonymous function. In case we want to define a function it will be invoked only once within the function do not waste the memory by giving a name to it, we can simply create anonymous function.

Syntax: function (){

}

➢ thisKeyword:

A predefined Key word been supported in JavaScript which refers the current object data.

In order to access objects data within its corresponding methods, we can access either by using object name or by using this keyword.

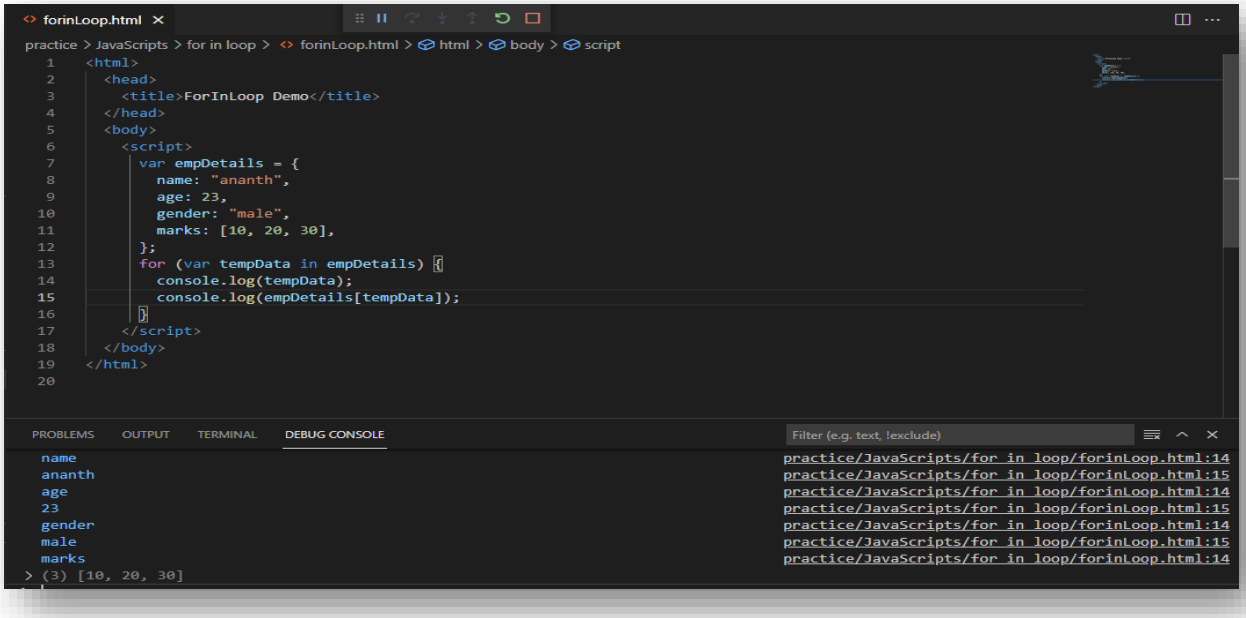
Ex:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
  </head>
  <body>
    <script type="text/javascript">
      var empdetails = {
        name: "raju",
        dept: "it",
        age: 20,
        basicsal: 25000,
        empsal: function () {
          this.hra = (15 / 100) * this.basicsal;
          console.log("HRA is : " + this.hra);
        },
      };
      empdetails.empsal();
    </script>
```

➢ for in loop Control Structure:

A Predefined looping Control Structure through which we could able to iterate over a **JSON** object.

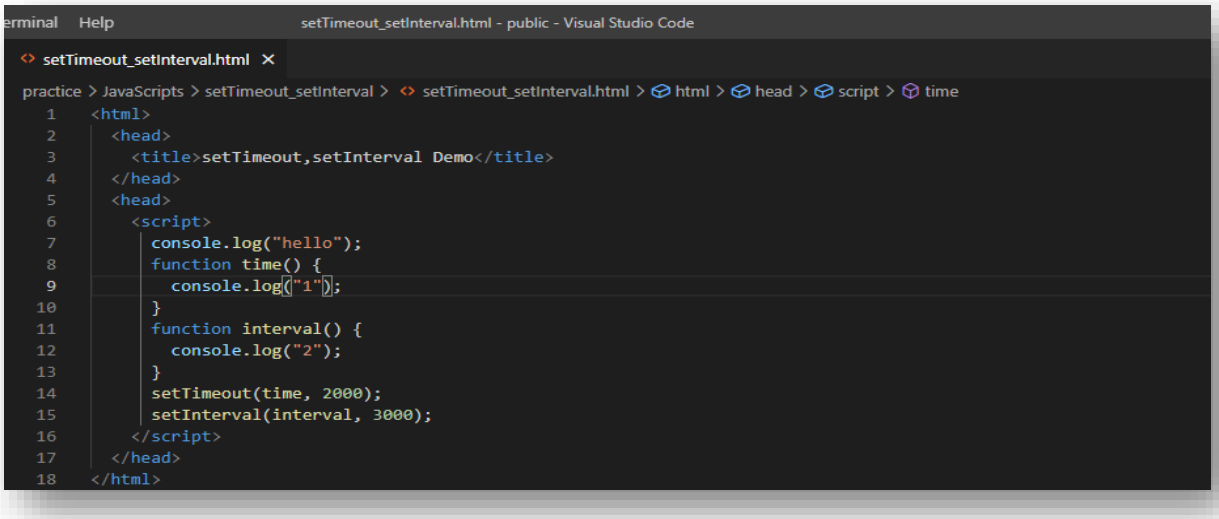
Ex:



➢ JavaScript 'setTimeout' and 'setInterval' methods:

A two predefined methods been supported in JavaScript using which we could able to invoke set of instructions not just when the controller reaches to it but, executes the set of instructions only after the provided interval time.

Syntax: setTimeout(<callbackmethod>,<delayvalue>);



- setInterval and setTimeout always works the same where the only difference is setTimeout invokes the function only once after the provided interval time whereas setInterval method invokes the call-back method repeatedly with a gap of provided interval time.

Note:

clearInterval () is a predefined method supported through which we could able to stop a running interval job.

Ex:

```
16          var count = 0;
17          var myInterval = setInterval(function () {
18            count++;
19            console.log("hello");
20            if (count == 4) {
21              clearInterval(myInterval);
22            }
23          }, 3000);
24          console.log("done");
25        </script>
26      </head>
27    </html>
```

DHTML – Dynamic HTML

- It is a powerful language comes with the predefined objects using which we could able to perform any type of **CRUD** operations on any DOM Elements on its corresponding CSS Property.
- Using JavaScript we could able to generate the complete HTML page with all **dynamic data** as and when required.
- It provides predefined methods and objects using which we could able to handle any type of events being generated on the page.
- It provides predefined methods and objects using which we could able to read, write or form on the HTML input elements.

Note:

'Document' and 'window' are two predefined object by default available in every web page is actually holds the complete DOM structure on which we can perform the CRUD operations.

➢ **'Document' and 'window':**

The two predefined objects been supported in JavaScript using through which we could perform any type of operation on any HTML Element of its corresponding CSS Properties.

Following are the different ways we could refer a DOM Elements on a page,

❋ **document.getElementById():**

Using which we could refer to a DOM Element on page using its unique ID.

❋ **document.getElementByClassName():**

Using which we could refer to a DOM Element on page using its name.

❋ **document.getElementByTagName():**

Using which we could refer to a DOM Element on page using its tag name.

❋ **document.querySelector():**

A method been supported in HTML5 using which we can refer to any DOM Element dynamically using JavaScript, may be through class, id or tag name.

Syntax: document.querySelector(#id / .class / tagName);

Following are the methods we can be applied on element reference,

❋ **element.setAttribute(“attributeName”,value):**

Using which we could able to set any attribute dynamically to a DOM Element.

Ex: spanTag.setAttribute('class', 'content');

spanTag.setAttribute("id", uid);

❋ **element.getAttribute(“attributeName”):**

Using which we can get corresponding attribute value of any DOM Element.

❋ **Element.style:**

Using which dynamically any CSS Property of any DOM Element can be retrieved or updated.

Ex: spanTag.style.border = '1px solid green';

spanTag.style.background = bgColor;

❋ **Element.appendchild(<element>):**

Using which we could able to append an element to an existing DOM Element page.

❋ **document.createElement(<tag name>:**

Using which we could able to create any DOM Element using JavaScript.

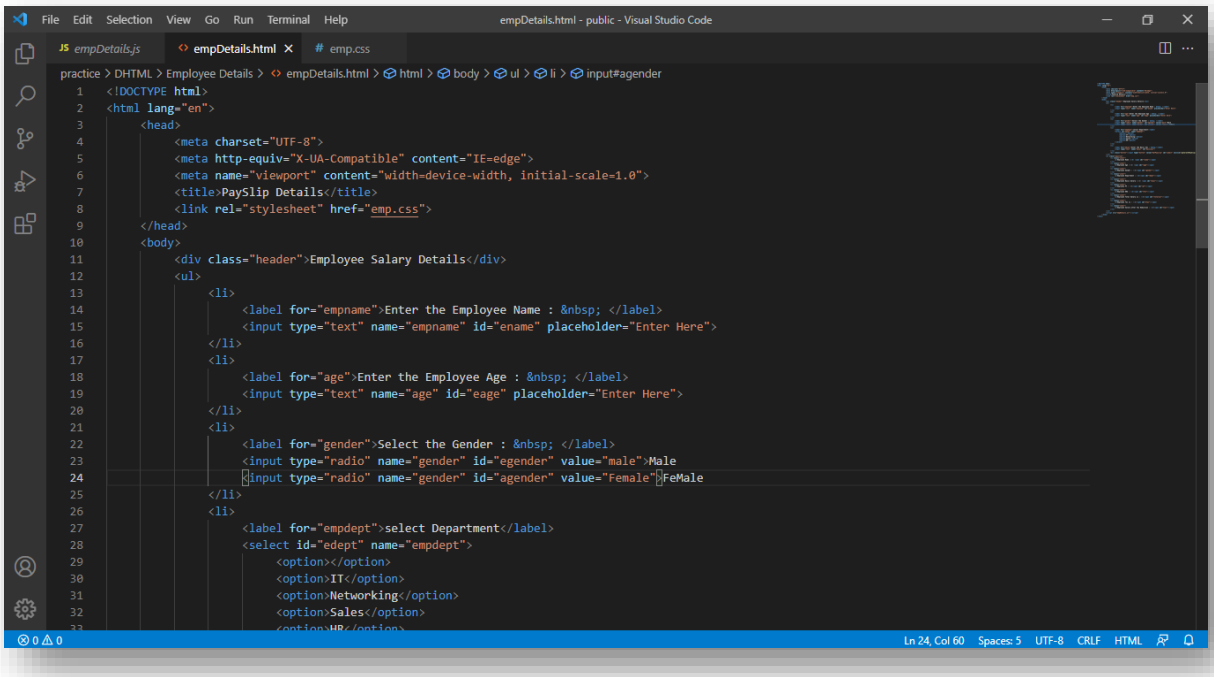
DHTML > DynamicTagandCSS.html > html > body > script

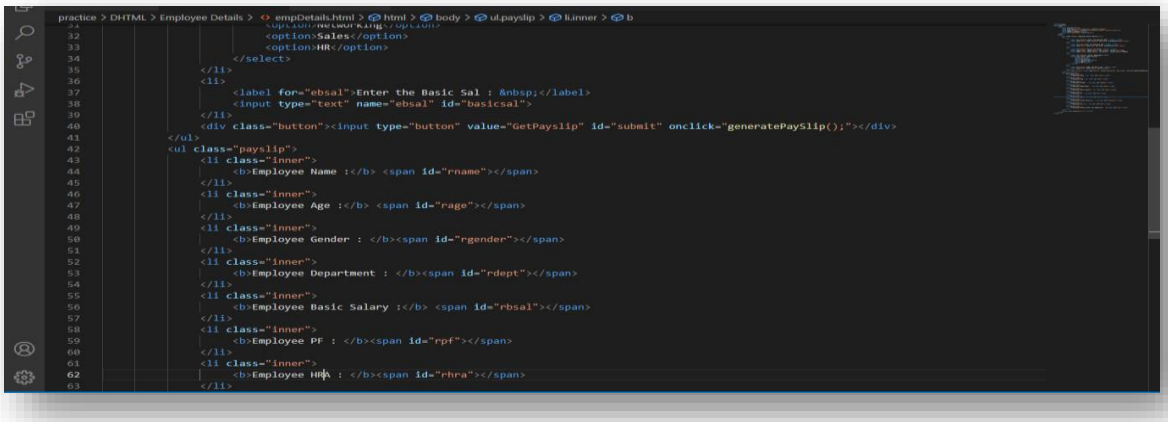
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
    </head>
    <body>
        <p class="demo">Hello Mr.Ananth_Atmakuri</p>
        <script>
            var demoTag = document.createElement("div");
            demoTag.setAttribute("class","divtag");
            demoTag.setAttribute("id","test");
            demoTag.style.border = "2px solid";
            demoTag.style.borderradius = "15px";
            demoTag.style.backgroundColor = "ghostwhite";
            demoTag.innerText = "Hello How Are You?"
            document.querySelector(".demo").append(demoTag);
        </script>
    </body>
</html>
```

➢ Reading values from Input Elements:

- JavaScript provides predefined methods and object through which we could able to read data and value from input elements.
- **element.value** which returns the values from input elements and text area.
- **element.option** which returns an array of options as object from a dropdown element.
- **element.selectedIndex** which returns the current selected index of dropdown element.
- **element.checked** which returns the true / false of current selected or non-selected state of checkbox.

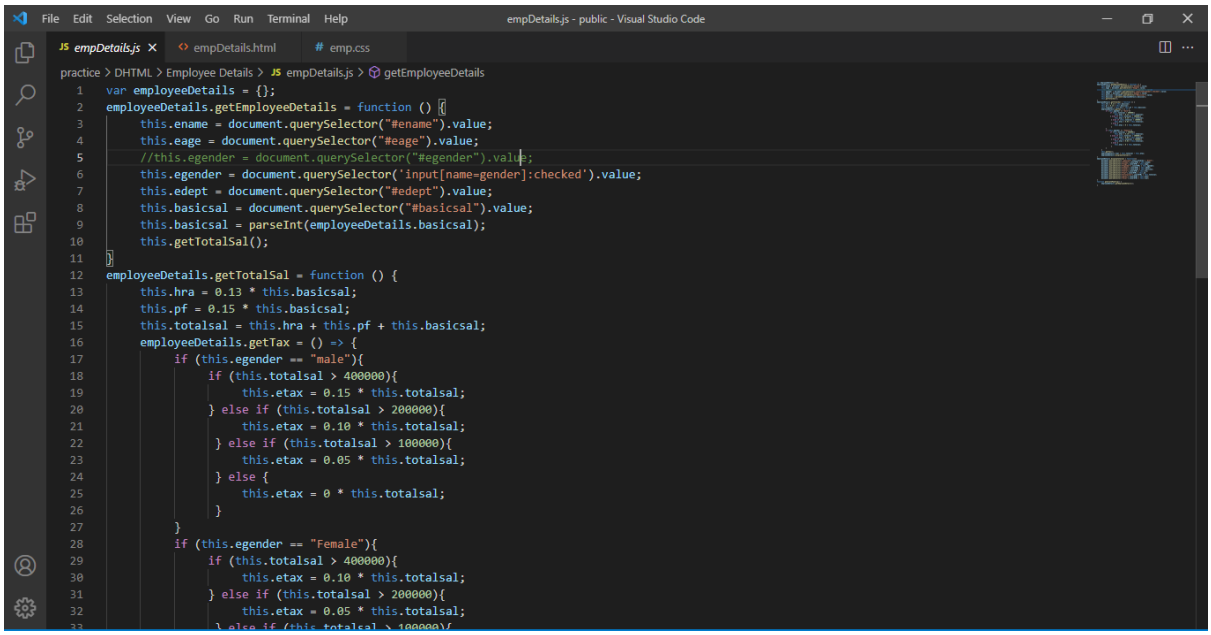
Ex: HTML Code



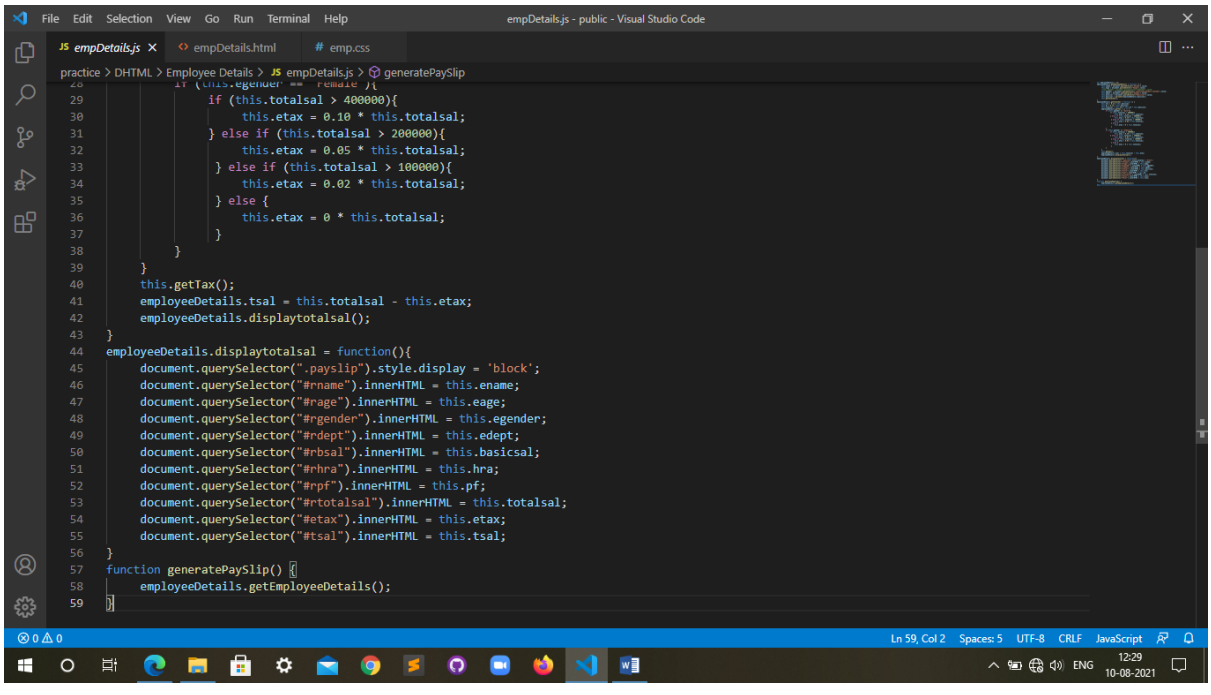


```
                </li>
                <li class="inner">
                    <b>Employee Total Salary is : </b><span id="rtotalsal"></span>
                </li>
                <li class="inner">
                    <b>Employee Tax is : </b><span id="etax"></span>
                </li>
                <li class="inner">
                    <b>Employee Salary after Tax Deduction : </b><span id="tsal"></span>
                </li>
            </ul>
            <script src="empDetails.js"></script>
        </body>
    </html>
```

- **JS Code**



```
var employeeDetails = {};
employeeDetails.getEmployeeDetails = function () {
    this.ename = document.querySelector("#ename").value;
    this.eage = document.querySelector("#eage").value;
    //this.egender = document.querySelector("#egender").value;
    this.egender = document.querySelector('input[name=gender]:checked').value;
    this.edept = document.querySelector("#edept").value;
    this.basicsal = document.querySelector("#basicsal").value;
    this.basicsal = parseInt(employeeDetails.basicsal);
    this.getTotalSal();
}
employeeDetails.getTotalSal = function () {
    this.hra = 0.13 * this.basicsal;
    this.pf = 0.15 * this.basicsal;
    this.totalsal = this.hra + this.pf + this.basicsal;
    employeeDetails.getTax = () => {
        if (this.egender == "male"){
            if (this.totalsal > 400000){
                this.etax = 0.15 * this.totalsal;
            } else if (this.totalsal > 200000){
                this.etax = 0.10 * this.totalsal;
            } else if (this.totalsal > 100000){
                this.etax = 0.05 * this.totalsal;
            } else {
                this.etax = 0 * this.totalsal;
            }
        }
        if (this.egender == "Female"){
            if (this.totalsal > 400000){
                this.etax = 0.10 * this.totalsal;
            } else if (this.totalsal > 200000){
                this.etax = 0.05 * this.totalsal;
```



```
            if (this.totalsal > 400000){
                this.etax = 0.10 * this.totalsal;
            } else if (this.totalsal > 200000){
                this.etax = 0.05 * this.totalsal;
            } else if (this.totalsal > 100000){
                this.etax = 0.02 * this.totalsal;
            } else {
                this.etax = 0 * this.totalsal;
            }
        }
    }
    this.getTax();
    employeeDetails.tsal = this.totalsal - this.etax;
    employeeDetails.displaytotalsal();
}
employeeDetails.displaytotalsal = function(){
    document.querySelector(".payslip").style.display = 'block';
    document.querySelector("#rname").innerHTML = this.ename;
    document.querySelector("#rage").innerHTML = this.eage;
    document.querySelector("#rgender").innerHTML = this.egender;
    document.querySelector("#rdept").innerHTML = this.edept;
    document.querySelector("#rbsal").innerHTML = this.basicsal;
    document.querySelector("#rhra").innerHTML = this.hra;
    document.querySelector("#rpf").innerHTML = this.pf;
    document.querySelector("#rtotalsal").innerHTML = this.totalsal;
    document.querySelector("#etax").innerHTML = this.etax;
    document.querySelector("#tsal").innerHTML = this.tsal;
}
function generatePaySlip() {
    employeeDetails.getEmployeeDetails();
}
```

➢ Event Handling:

❋ Event:

Any action been performed on a page is called an event.

❋ The process of performing a particular job based on a particular action been generated on a page is called an Event Handling.

❋ Following are the types of action might get generated on a page,

- Click - dbClick - mouseover
- Drag - drop - keypress
- Keydown - focus - blum

❋ Following are the two ways we could able to implement event handling,

- Static Event Handling
- Dynamic Event Handling

❋ Static Event Handling:

In this way, using HTML predefined static event attributes, we statistically specify the particular JavaScript function need to be invoked when a particular action happens on a page.

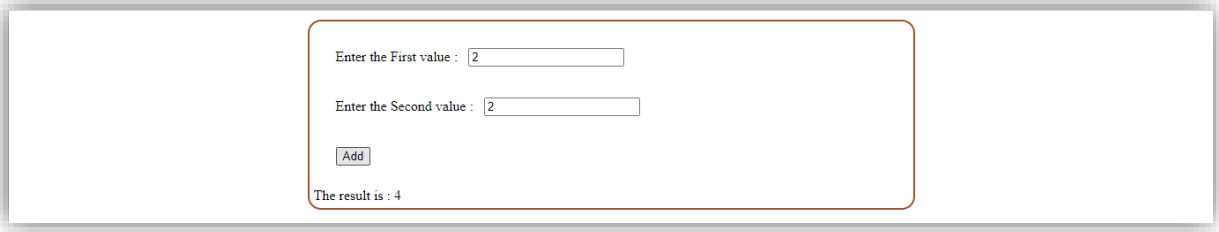
Following are the pre-defined HTML Attributes,

- onclick -onmouseover -onfocus -onchange -onblur

-onkeyup -onkeydown -onmouseout -ondbclick

**Ex: **

```
<body>
    <ul>
        <li>
            <label for="fval">Enter the First value :   </label>
            <input type = "text" name = "fval" id = "fVal" required = "fval">
        </li>
        <li>
            <label for="sval">Enter the Second value :   </label>
            <input type = "text" name = "sval" id = "sVal" required = "sval">
        </li>
        <li>
            <input type="button" value = "Add"  onclick="addition();">
        </li>
        <span id = "res"></span>
    </ul>
    <script>
        function addition() {
            var firstVal = document.querySelector("#fVal").value;
            firstVal = parseInt(firstVal);
            var secondVal = document.querySelector("#sVal").value;
            secondVal = parseInt(secondVal);
            var result = firstVal + secondVal;
            result = document.querySelector("#res").innerHTML = "The result is : " + result;
        }
    </script>
</body>
```



✵ Dynamic Event Handling:

- JavaScript provides a feature through which we can dynamically specify what method need to be invoked when a particular action happens on a DOM Element.
- **"addEventListener"** It is a predefined method using which we can handle action been predefined on a page dynamically.

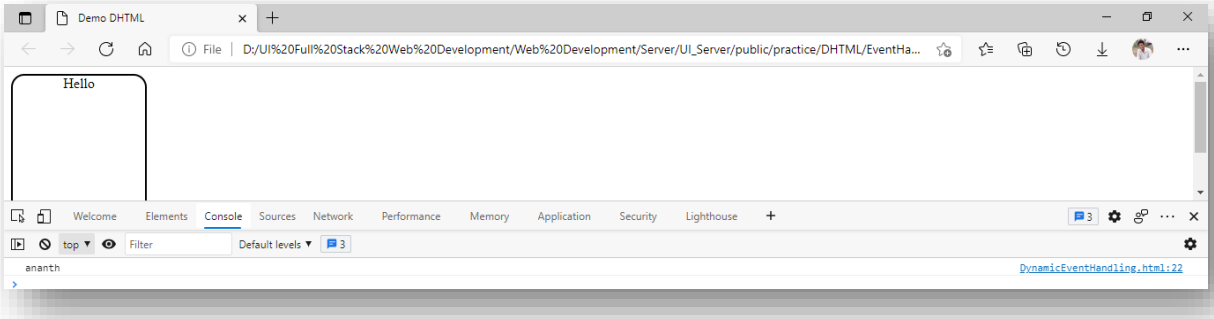
  Syntax: element.addEventlistener("typeofEvent",<callbackmethod>);

- This method attaches an event handler to an element without corresponding existing event handling.
- You can add many event handlers to any element.

Ex:

practice > DHTML > EventHandling > DynamicEventHandling.html > html > head > style > .sample

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Demo DHTML</title>
        <style>
            .sample {
                border: 2px solid;
                border-radius: 15px;
                text-align: center;
                width: 150px;
                height: 200px;
            }
        </style>
    </head>
    <body>
        <div class="sample">Hello</div>
        <script>
            document.querySelector('.sample').addEventListener('click',()=>{
                console.log('ananth');
            })
        </script>
    </body>
</html>
```



➢ Event Bubbling and Capturing.

➢ Inheritance:

The process of defining an object and accessing the data of it (data members and data functions) within another object is called inheritance.

Inheritance is a feature in which one class inherits the property of another class. A class which inherits the property is called a **derived class** or **subclass** or **child class** and from which derived class inherits property is called as **a base class** or **parent class**.

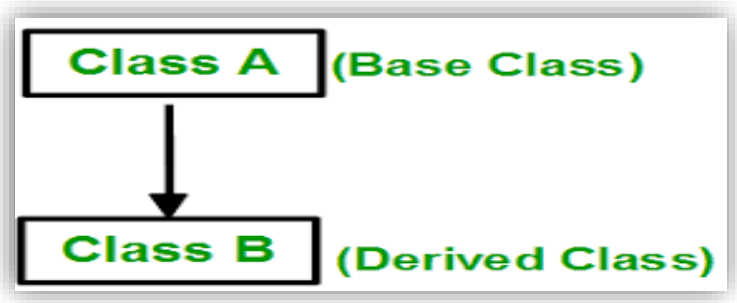
Inheritance makes **reusability** of code, which avoids rewriting the existing code again and again. A class can be driven from one class or from multiple base classes.

JavaScript supports two types of inheritance,

I. Single level Inheritance.
II. Multilevel Inheritance.

❋ Single level Inheritance:

Single Level inheritance - A class inherits properties from a single class. For example, Class B inherits Class A.

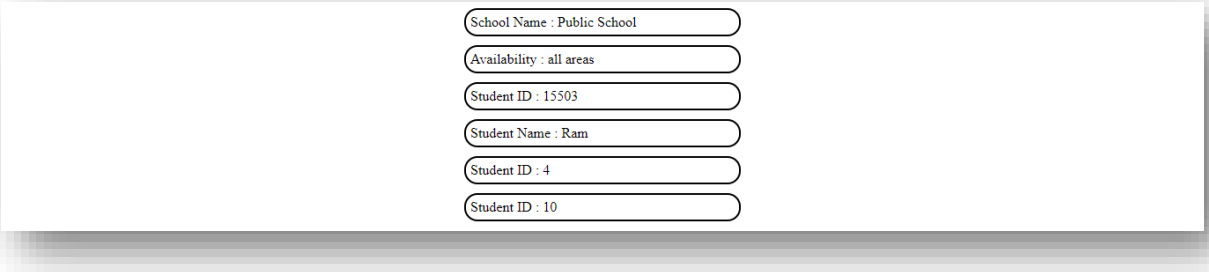


Ex:

```
1  var schoolDetails = {
2      schoolName : 'Public School',
3      area : 'all areas'
4  }
5
6  var studentDetails = {
7      sid: 15503,
8      name: 'Ram',
9      Age: 10,
10     class : 4,
11     displaystudentDetails: function() {
12         document.querySelector('#school').innerHTML = 'School Name : ' + this.schoolName;
13         document.querySelector('#state').innerHTML = 'Availability : ' + this.area;
14         document.querySelector('#sid').innerHTML = 'Student ID : ' + this.sid;
15         document.querySelector('#Name').innerHTML = 'Student Name : ' + this.name;
16         document.querySelector('#class').innerHTML = 'Student ID : ' + this.class;
17         document.querySelector('#age').innerHTML = 'Student ID : ' + this.Age;
18     }
19 };
20
21 studentDetails.__proto__ = schoolDetails;
22 studentDetails.displaystudentDetails();
```

Here schoolDetails object is parent i.e. base class

Here studentDetails is child class or derived class



- **Note:**

  Due to data ambiguity issues in JavaScript does not support Multiple Inheritance.

  Following are the different ways between types of object inheritance can be implemented.

- Inheritance between two static objects.
- Inheritance between static object and dynamic object.
- Inheritance between two dynamic objects.

❋ **Inheritance between two static objects:**

- Every static object been created in JavaScript holds a predefined property by default "**__proto__**".
- Using "**__proto__**" we could able to extend the behaviour of any object to access another existing object with properties and behaviour.
- The main object from which data gets derived is called **parent object** or **base object.**
- The object which is accessing the data or got derived from the parent object is called **child object** or **derived object.**
- For Example Refer **Single Level Inheritance.**

❋ **Inheritance between static object and dynamic object:**

- **"object.create()"** is a predefined way to create a dynamic object by inheriting data from a static object.
  Syntax: object.create(<existing static object>);
  Ex: var data = {

  }
  var childobj = object.create(data);

Here **childobj** holds its own properties along with it can point to existing data of "**data**" object.

```
var schoolData = {
    schoolName: 'Abc School',
    schoolLocation: 'Hyderabad',
    schoolRating: '4/5'
}

function registerStudent() {
    var studentData = Object.create(schoolData);
    // var studentData = {};
    studentData.name = document.querySelector("#uname").value;
    studentData.age = document.querySelector("#uage").value;
    studentData.class = document.querySelector("#uClass").value;
    studentData.address = document.querySelector("#uAddress").value;

    var ulTag = document.createElement("ul");

    for (var key in studentData) {
        var li = document.createElement('li');
        li.innerHTML = key + ' : ' + studentData[key];
        ulTag.append(li);
    }
    document.querySelector(".detilsContaianer").append(ulTag);
}
```

➢ JavaScript Class:

Class indicates or represents a predefined structure of an object so that we can create any number of objects having same structure but with different data until **ECMA5**.Classes we are not directly supported in JavaScript, using functions we could indirectly get the feature of Class.

From **ECMA6,** Class is a predefined Keyword through which we could able to create classes directly.

✧ ECMA5 Class Syntax:

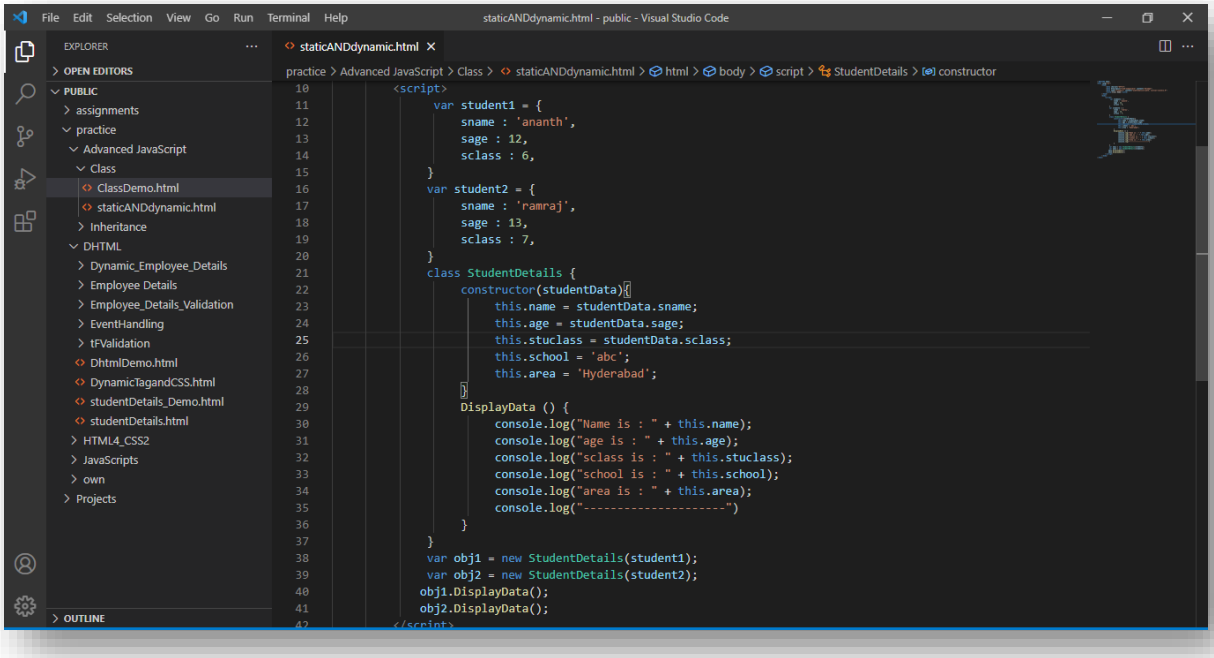
```
function <class name>(<optional params>){
        this.key = '..';
        this.key2='..';
        ……..
        this.method = function() {
        ……
        }
}
```

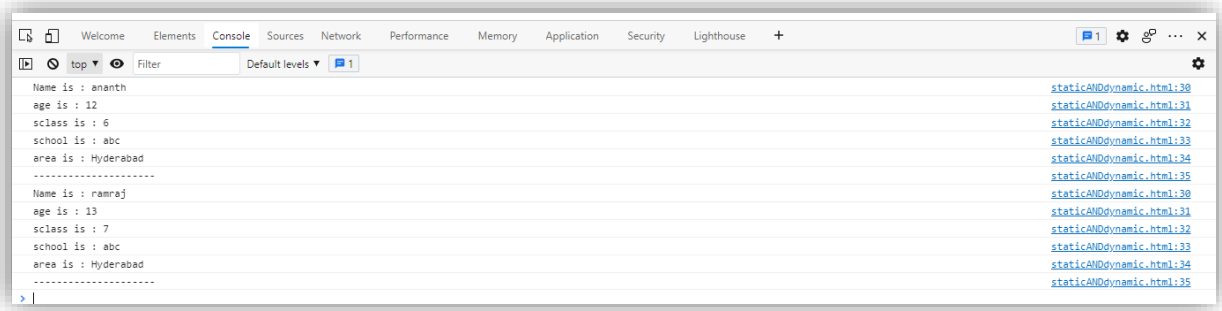
✧ ECMA-6:

```
Class ClassName {
        constructor(optionalparams) {
                this.key = '..';
        this.key2='..';
        ……..
```

```
        }
        methodName() {
            ……..
        }
    }
```

Ex:





- In Class first the **Constructor** is executed.

➢ **<u>inherit from two Classes:</u>**

The **extends** keyword is used to create a child class of another class (parent).
The child class inherits all the methods from another class.
Inheritance is useful for code reusability: reuse properties and methods of an existing class when you create a new class.

The super() method refers to the parent class. By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

```
Syntax : class Parent {
        Constructor() {
                .......
        }
.........
}
class child extends parent {
                constructor () {
                        super();
                        .........
                }
}
```

Note:

Super() keyword should be declared in constructor in the first line itself in the child class.

Ex:

```
<script type="text/javascript">
    class Parent {
        constructor(obj1) {
            this.a = 10;
        }
        displayParent () {
            console.log("The Value of a is : " + this.a);
        }
    }
    class Child extends Parent {
        constructor(obj2){
            super();
            this.b = 20;
            this.result = this.a + this.b;
        }
        displayChild() {
            console.log("The Value of b is : " + this.b);
            console.log("The Result is : " + this.result);
        }
    }
    var obj1 = new Parent(obj1);
    var obj2 = new Child(obj2);
    obj1.displayParent();
    obj2.displayChild();
</script>
</body>
</html>
```

```
Elements  Console  Sources  Network  Performance  Memory  Application  Security  Lighthouse
top  Filter  Default levels  3
The Value of a is : 10                    Inheritance2Classes.html:16
The Value of b is : 20                    Inheritance2Classes.html:26
The Result is : 30                        Inheritance2Classes.html:27
```

➢ Exception Handling:

- **Error/Exception:**

A set of instructions or a line of instructions making the execution of application to stop abruptly.

The errors or exceptions might get raised while compiling the code or while executing the code.

I. Runtime Exception.
II. Compile Time Exception.

- **Compile Time Exception or Syntactical Exception:**

The errors which gets raised while we compile the code is called compile time errors.

Ex: Syntactical Errors.

JavaScript does not provide any way to handle compile time errors these errors are for sure need to be fixed to go to the further execution.

- **Run-Time Exceptions:**

The exceptions which gets raised while executing the code is called the runtime exceptions or the runtime errors.

Ex:

1. Array index out of bonds.

2. File not found exception.

3. Referring a DOM Element which does not exists.

In general when a runtime error gets raised the flow of execution gets stopped abruptly without going to the further execution.

JavaScript provide a feature of handling exception through which we could able to handle the exceptions being raised and continue the execution flow without stop.

Following are the predefined keyword through which we could able to handle Exceptions,

i. Try
ii. Catch
iii. Finally
iv. Throws.

Using **try, catch** statement we could able to handle exceptions get raised at the runtime.

- **Try Block:**

The set of instructions in which there is a chance of getting runtime errors has to be placed under the **TRY** block.

- **Catch Block:**

The set of instructions through which we could able to handle the exception has to be placed under the **Catch** block.

- **Catch** block is the immediate block should be placed after the **try** block.
- We cannot overload catch method in JavaScript, a single **try** block should have corresponding single catch method.
- We cannot invoke the **catch** method manually. It gets invoked automatically when there is an exception under the **try** block.
- Set of instructions through which we handle the error been raised at **try** block as to be placed under **catch** method.
- **Catch** method automatically gets involved holds an exception or error object with extra information about the current error been generated.

Syntax: try {

::///set of instructions in which there is a chance of getting runtime errors

} catch (error obj) {

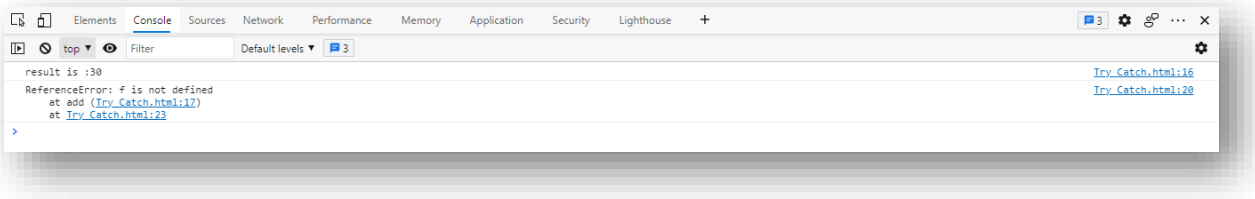
….

// set of instructions to handle error been raised.

}

Ex:

```
10      <script type="text/JavaScript">
11          var add = () => {
12              var a = 10;
13              var b = 20;
14              try {
15                  var c = a+b;
16                  console.log("result is :" + c);
17                  d = a + f + c;
18                  console.log(d);
19              } catch (error) {
20                  console.log(error);
21              }
22          }
23          add();
24      </script>
```



➢ Finally Block:

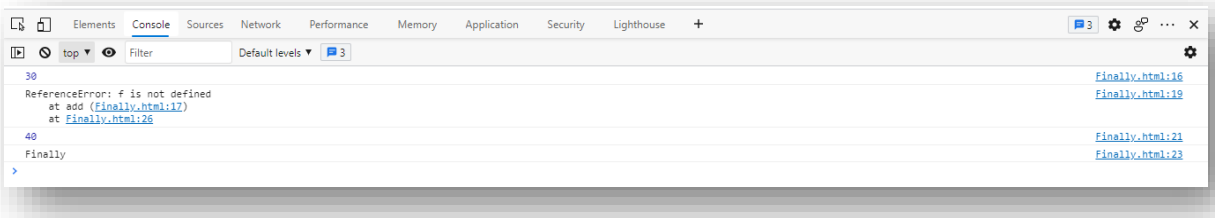
An optional block can be placed after the catch block which gets executed for sure irrelevant of whether there is a exception being raised within the try block or not.

- The set of instructions which for sure need to be executed irrelevant of exception been raised or not with in the try block, has to be placed under the finally block.

  Syntax: try {
  ….. // Set of instructions in which there is a chance of
  ….. // getting error.
  } catch (error) {
  ….. // Code to handle error
  } finally {
  …… // set of instructions need to executed for sure,
  …… // irrelevant of whether there is an exception
  …… // raised or not.
  };

Ex:

```
        <script type="text/JavaScript">
            var add = () => {
                var a = 10;
                var b = 20;
                try {
                    var c = a + b;
                    console.log(c);
                    c = a+b+f+c;
                } catch (error) {
                    console.log(error);
                    b = a + c;
                    console.log(b);
                } finally {
                    console.log("Finally");
                }
            }
            add();
        </script>
    </body>
</html>
```

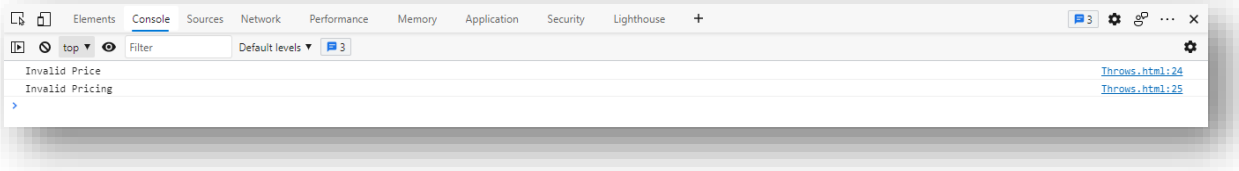


➢ Handling User Defined Exceptions:

JavaScript provides a feature of handling user-defined exceptions using **"throws"** keyword. Any time the controller reaches to the throw statement within the **try** block, it automatically treats that as an error an invokes the corresponding catch block.

Ex:

```
10          <script>
11              var pdprice = prompt("Enter the Value of a : ");
12              var tax = prompt("Enter the value of b : ");
13              pdprice = parseInt(a=pdprice);
14              tax = parseInt(tax);
15              var total = pdprice + tax;
16              try {
17                  if (total > 20){
18                      console.log("valid price");
19                  }
20                  if (total < 20){
21                      throw "Invalid Price"
22                  }
23              } catch(error) {
24                  console.log(error);
25                  console.log("Invalid Pricing")
26              }
27          </script>
```



➢ Closures:

Closure is a self-invoked functions gets invoked automatically once the controller reaches to it.

- Using closure, we could able to find set of JavaScript instructions as an individual module.
- The set of instructions within a closure module becomes a private data, and cannot be accessed outside of the closure.
- Within a closure we can able to find set of JavaScript instructions like set of variable methods, objects, classes etc…
- The data within a closure cannot be accessible outside of it even within the same page, because it adds accessibility security to that data.

Syntax: (function() {

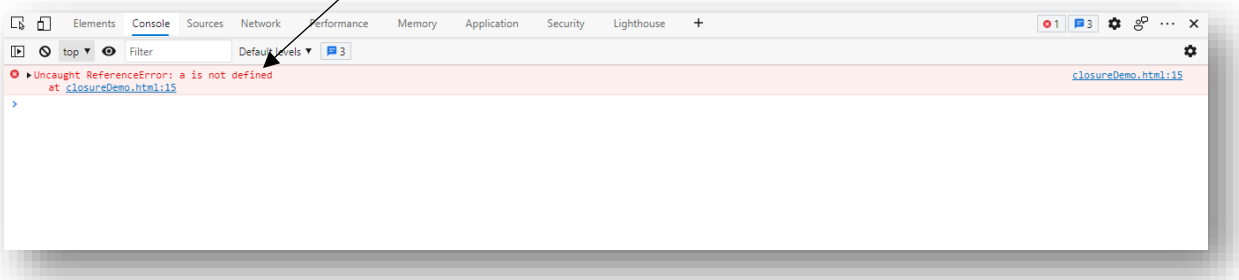
……..//set of instructions.

})();

Ex:

```
10      <script>
11        ( ()=> {
12            var a = 10;
13            var b = 20;
14        })();
15        var c = a + b;
16        console.log(c);
17      </script>
```

This is the closure block and we cannot use the variables/methods that are declared in it



➢ Accessing data in closure outside of it:

Steps to be followed to access the closure data,

- Create and assign a variable to the closure.
- Return the data which is needed to access outside of the closure as like an object.
- Using closure name, we could able to access closure returned object data outside of the closure.

Note:

Not all the data within the closure can be accessible outside of it, only the data which is returned from the closure can be accessible through closure name.

Syntax: **var closureName = (( ) => {**

…….// set of instructions

return {

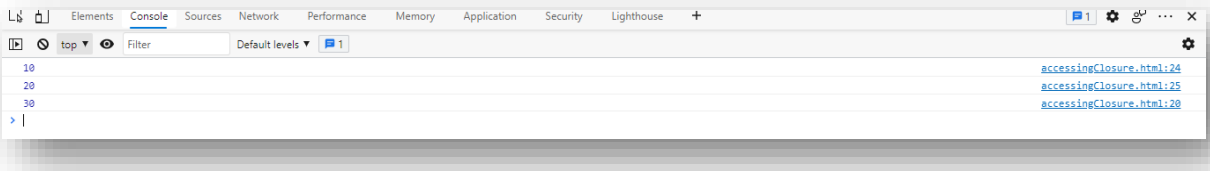
….// data to be accessed outside

})( );

Ex:

```
10      <script>
11          var sample = ( () => {
12              var a = 10;
13              var b = 20;
14
15              return {
16                  firstval : a,
17                  secondval : b,
18                  result : () => {
19                      var result = a + b;
20                      console.log(result);
21                  }
22              }
23          })();
24          console.log(sample.firstval);
25          console.log(sample.secondval);
26          sample.result();
27      </script>
```

By using this we are accessing the data within the closure.



JQUERY

➢ Introduction to jQuery:

A predefined JavaScript library developed by Mr. John Resig in the Year 2006, comes with predefined method through which the DOM Operations performed within the page can be simplified.

- It supports predefined methods through which we can add, remove (or) update any CSS Property of any HTML Element dynamically.
- It provides a set of predefined methods through which we can add, remove or update any html element dynamically.
- It provides a set of predefined methods through which we can handle dynamic events generated on page.
- It provides set of predefined methods and objects through which ajax communication within the page implemented easily.
- It supports methods of implement animations to the DOM element.
- It provides set of predefined methods through which we could able to refer existing elements on the page in different ways.
- It provides different set of selectors and travelling methods.
- 3.6.0 is the current and latest version of jQuery, https://jquery.com is the website where we can download, even provides API documentation of jQuery library.
- jQuery library is of two types
    I. Compressed version
    II. Uncompressed version

Links for jQuery library

I. https://code.jquery.com/jquery-3.6.0.js //uncompressed
II. https://code.jquery.com/jquery-3.6.0.min.js //compressed.

Ex:

```
        <style>
            .borders {
                border: 2px solid;
                border-radius: 10px;
                padding: 10px;
                margin: 10px;
                width: 50px;
            }
            .changes {
                color: azure;
                background-color: black;
                font-weight: bold;
            }
        </style>
        <script src="../jQueryCompressed.js"></script>
    </head>
    <body>
        <div class="borders" id="test">Hello jQuery</div>
        <input type="button" value="Change" onclick="change();">
        <script>
            var change = () => {
                $("#test").addClass("changes");
            }
        </script>
```

➢ Minified and non-minified code:

While loading the content on the browser, it is recommended that to have less number of files so that browser takes less time to load them which in turn increases performance of the page.

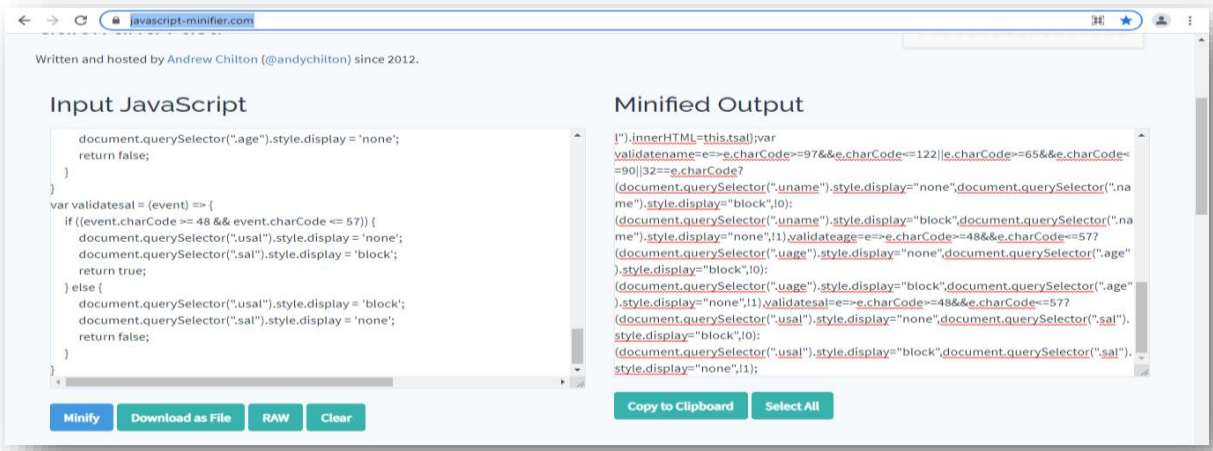
- It is recommended every JavaScript file of a webpage has to go under minification process before it gets deployed on production environment.
- Minification is the process of reducing the total JavaScript code been written.

Following are the steps to be followed in minifying code:

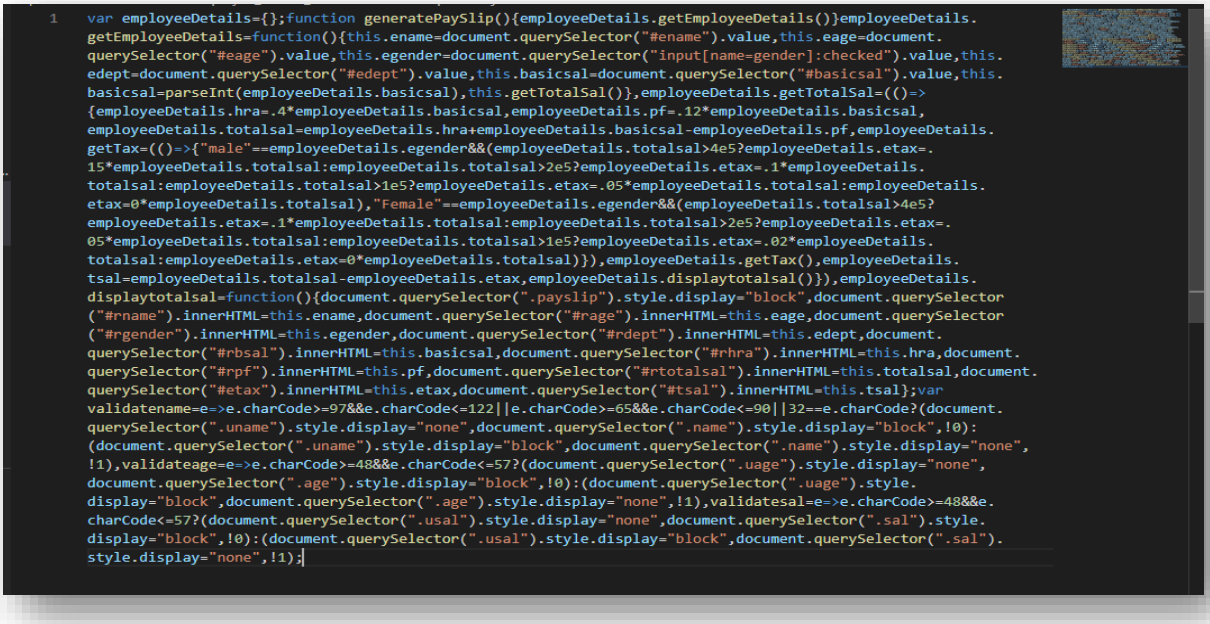
- It removes all the extra spaces within the code.
- Replaces all the variable names with smaller ones.
- Optimises the logic as much as possible.
- Removes unnecessary commands and descriptions within the file.

➢ Steps the minify the JS code:

- Select the JavaScript file which you want to minify.
- Visit https://javascript-minifier.com
- Select the code and paste and press minify,



- This is the minified code,



➢ jQuery selectors and methods:

$("*") - To get any element with the dom

$("#abc") - Returns the reference of element with id abc

$(".abc") - Returns the reference of element with class abc

$("div.abc") -> Returns the reference of all div tags having the class as abc

$("li:first") -> Returns the reference of li tag which is in first child position

$("input[type=text]") -> Returns reference of all input elements having type as text

$("a[target!="_blank"]") -> Returns reference of all "a" tags which are not having target as "_blank"

$("tr:even") -> Returns reference of all "tr" tags which are in even position

$("div:odd") -> Returns reference of all "div" tags which are in odd position

$("div > span") -> Returns reference of all span tags which all are direct childrens of div tag

$("li:eq(2)") -> Returns reference of li tag which is in second position

$("span:first-of-type") -> selects all elements that are the first among siblings of the same element name

$("div:gt(3)") -> selects elements at an index greater than index within the matched set

$("div:has(p)") -> REturns div tag which as atleast one p tag

$("div:hidden") -> Returns reference of div tag which is in hidden state

$("span:last-of-type") -> Returns span which is last sbling

$("div, .abc, span, #pqr") -> Returns multipel selectrs

$("input:disabled") -> Returns input element with disabled state

$("input:checked") -> Returns check box element which is in checked state.

etc..

- **Jquery Predefined methods:**

Following are list of predefined methods been supported in jquery can be applied on DOM elements.

.addClass() - to add single or multiple classes to an element

.removeClass("classname") - To remove a classname

.hasClass("classname") - Returns true /false based on, if element is having the provided class or not.

.after() -> To add an element as siblint after the existing element

eg:

```
$("div").after("<p>test</p>");
```

.before() -> To add an element before the existing element

eg:

```
$("div").before("<p>test</p>");
```

.append() -> to append an element to existing element

.attr() - To set or get any attribute of an element

.css() - > To set single or multiple css properties to elements

eg:

```
$("#abc").css("color", "green");
$(".container").css({
    color: 'blue',
    'font-size': '20px'
})
```

.val() -> To get or set value from input elements

.removeAttr() -> To remove and existing attribute from element

.toggleClass() -> to toggle class from existing element

.html("html text") -> adds required content as html content with in an element

.text("text content") -> adds required content as text content with in an element

.show() -> to show an hidden element

.hide() -> To hide an visible element

etc.

- **$(document).ready():**

  A page can't be manipulated safely until the document is "ready." jQuery detects this state of readiness for you. Code included inside $(document).ready() will only run once the page Document Object Model (DOM) is ready for JavaScript code to execute. Code included inside $(

window).on("load", function() { ... }) will run once the entire page (images or iframes), not just the DOM, is ready.

Ex:
```
$( document ).ready(function() {
   console.log( "ready!" );
    });
```
Ex2:

```
Line wrap
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Demonstrating Jquery MEthods</title>
        <script src="../jquery.js"></script>
        <script src="readyAndLoad.js"></script>
    </head>
    <body>
        <div id="container">
            Click here
        </div>
    </body>
</html>
```

JS Code:

```
console.log("heelo");

$(document).ready(function(){
    document.querySelector("#container").addEventListener('click', () => {
        console.log("user clicked");
    });
});
```

➢ jQuery traversing Methods:

following are the predefined methods through which we could traverse between DOM Elements,

- **parent():** Returns the immediate parent of any DOM Element.

```
        <style type="text/css">
            .text {
                border: 2px solid;
                color: aliceblue;
                background-color: black;
                list-style: none;
            }
        </style>
    </head>
    <body>
        <ul>
            <li>one</li>
            <li>two</li>
            <li><button>three</button></li>
            <li>four</li>
            <li>five</li>
        </ul>
        <script>
            $("li").on("click", () => {
                $(event.target).parent().addClass("text");
                $(event.target).css("border-radius","15px");
            })
        </script>
```

- **parents():** Returns all parents until HTML tag.
- **parentsUntil("pattern"):** Returns all the parents until it finds a parent with given pattern.

```
<script>
    $("li").on("click", () => {
        // $(event.target).parent().addClass("text");
        // $(event.target).parents().addClass("text");
        $(event.target).parentsUntil("body").addClass("text")
        $(event.target).css("border-radius","15px");
    })
</script>
```

➢ Traversing siblings:

- **siblings():** Returns the list of all siblings of any DOM Element.
- **next():** Returns the reference of next sibling element.
- **prev():** Returns the reference of Previous sibling element.
- **prevAll():** Returns the reference of all Previous sibling element.
- **nextAll():** Returns the reference of all Next sibling element.
- **nextUntil("Pattern"):** Return all the next siblings until the provided pattern.
- **previousUntil("Pattern"):** Return all the previous siblings until the provided pattern.

```
a launch.json file.

Show all automatic debug configurations.

<body>
    <input type="button" value="action" onclick="doaction()">
    <ul>
        <li>one</li>
        <li>two</li>
        <li id="3">Three</li>
        <li>four</li>
        <li>Five</li>
        <li>Six</li>
    </ul>
    <script>
        function doaction() {
            //$("#3").siblings().addClass("apply");
            //$("#3").next().addClass("apply");
            //$("#3").nextAll().addClass("apply");
            //$("#3").nextUntil("ul").addClass("apply");
            //$("#3").prev().addClass("apply");
            //$("#3").prevAll().addClass("apply");
            $("#3").prevUntil("body").addClass("apply");
        }
    </script>
</body>
```

➢ Traversing Children:

- **children():** Return the children of any DOM Element.
- **find(*):** Return the every children of a DOM Element.

➢ Adding Event Handling to DOM Element through jQuery Methods:

Following are the predefined methods provided by the jQuery which can be applied on DOM Elements to handle events.

element.click();

element.focus();

element.change();

element.blur();

element.mouseover();

Ex: $("abc").click(callingMethod);

- **"on"** is a method been provided by jQuery through which we can add event handling dynamically to DOM Elements.
  Syntax: $("abc").on("<typeofevent>", <callbackfunction()> {
  …….. // lines to be executed;
  });
- Using **"on"** we could able to add multiple events at a time to single DOM Element.
  Syntax: $("abc").on("<typeofevent>",
  <callbackfunction()> {
  …….. // lines to be executed;
  },
  <callbackfunction2()> {
  …….. // lines to be executed;
  });

```
<script>
    $("li").on("click", () => {
        // $(event.target).parent().addClass("text");
        // $(event.target).parents().addClass("text");
        $(event.target).parentsUntil("body").addClass("text")
        $(event.target).css("border-radius","15px");
    })
</script>
```

➢ jQuery Animation:

Adding animations to DOM Elements using jQuery methods following are the different ways we could add animation to DOM Elements.

1. Sliding
2. Fade
3. Hide/Show/Toggle

- **Hide/Show/Toggle:**

```
<div id="demo">Hello World</div>
        <input type="button" id="btn" value="show">
        <input type="button" id="btn1" value="Hide">
        <input type="button" id="Toggle" value="Toggle">
        <script type="text/JavaScript">
            $("#btn").on("click",() => {
                $("#demo").addClass("container").show();
            });
            $("#btn1").on("click",() => {
                $("#demo").hide();
            });
            $("#Toggle").on("click",() => {
                $("#demo").addClass("container").toggle();
            });
```

- **Fade:**

```
<input type="button" value="Fade in" onclick="doAction('in')">
        <input type="button" value="Fade out" onclick="doAction('out')">
        <input type="button" value="Toggle" onclick="doAction('tog')">
        <div class="container">
            CONTENT
        </div>
        <script>
            var doAction = (type) => {
                switch (type) {
                    case 'in':
                        $(".container").fadeIn(8000);
                        break;
                    case 'out':
                        $(".container").fadeOut(8000);
                        break;
                    case 'tog':
                        $(".container").fadeToggle(4000);
                        break;
                }
            }
        </script>
```

- **Sliding:**

```
<script>
            var doAction = (type) => {
                switch (type) {
                    case 'up':
                        $(".container").slideUp(8000);
                        break;
                    case 'down':
                        $(".container").slideDown(8000);
                        break;
                    case 'tog':
                        $(".container").slideToggle(4000);
                        break;
                }
            }
        </script>
```

- **Master Animation Method:**

Animate method is supported in jQuery, which can take multiple CSS Properties and add to DOM Elements with provided delay time.

Syntax:

```
element.animate({
    cssprop1 : value;
    cssprop2 : value;
    .....
    .....
},delaytime);
```

Ex:

```
<script>
          var doAction = (type) => {
              //$(".container").css("left", "300px");
              $(".container").animate({
                  left: '800px',
                  height: '200px',
                  width: '200px'
              }, 3000, function() {
                  $(".container").animate({left: '10px'}, 3000);
              });
          }
      </script>
```

AJAX

- **Web Service:**

Set of instructions gets executed at the server side which are capable of considering a request from client side, process the request accordingly and response back is called webservice.

- In a single server, it can run any number of web Services.
- Every Web Service within the server gets identified with a unique **URL**.
- All the Web Services within the server runs independent to each other.

Ex: Sample webservices URL's

https://www.amazon.in/ref=nav_logo

- **AJAX:**

The process of creating a request from client side to a server through a webservice **URL** in order to indirectly process operations on database is called **AJAX**.

- **Synchronous and Asynchronous Communication:**

While interacting to the server from the client, at a time we could able to create any number of requests, while communicating to the server, the communication type could be synchronous or asynchronous communication.

- **Synchronous Communication:**

In this type, once the request been sent to the server, from client side it doesn't execute further instructions until there is a response from the server.

- **Asynchronous Communication:**

In this type, once the request been sent to the server, client doesn't wait for its response and it continues executing further instructions. Call back function holds the set of instructions and executes automatically once there is response from the server.

- **Specifying the data type while Communicating:**

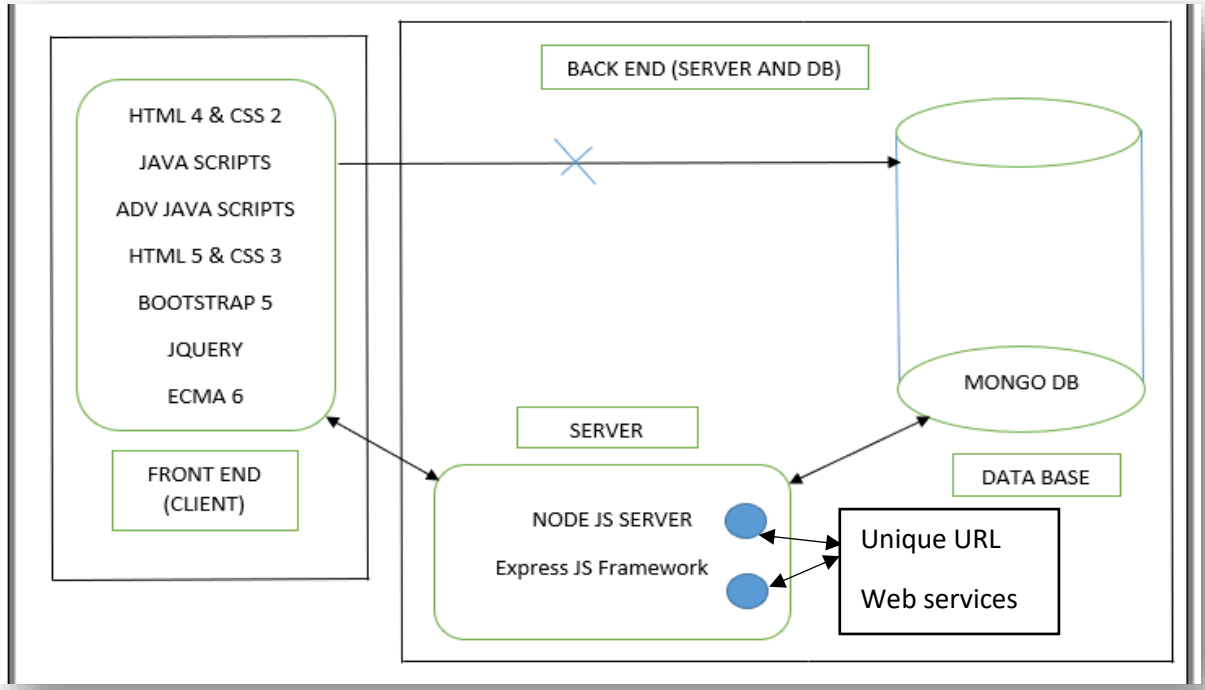
While interacting to the server, through a webserver, we need to specify the, type of the data format in which communication happens.

Following are the different data types in which communication can be done,

-Json -String -XML -RSS etc…

Among all the datatypes **Json** is mostly used and recommended **Data Type**.

➢ AJAX:



➢ Creating a AJAX call through jQuery AJAX Method:

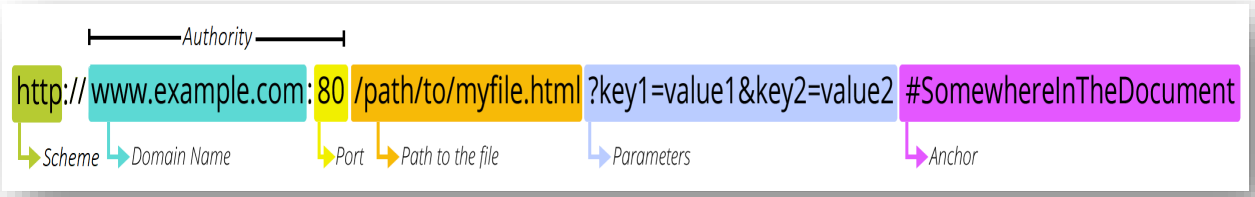
"$.ajax()" is a predefined method supported in jQuery using which we could able to create an AJAX call to a specific webservice from client side.

It takes an object as a parameter with the following Properties,

❋ URL:

URL stands for **Uniform Resource Locator** through which we can specify the data, type of data through which communication happens (XML/JSON/STRING etc…). A URL is nothing more than the address of a given unique resource on the Web. In theory, each valid URL points to a unique resource. Such resources can be an HTML page, a CSS document, an image, etc. In practice, there are some exceptions, the most common being a URL pointing to a resource that no longer exists or that has moved. As the resource represented by the URL and the URL itself are handled by the Web server, it is up to the owner of the web server to carefully manage that resource and its associated URL.

Ex: `https://developer.mozilla.org/en-US/search?q=URL`



- **Method:**

  Through which we need to specify the type of communication Secured/Non-Secured (Get/Post).

- **async:**

  Takes a Boolean value, through which we can specify whether the call is synchronous or asynchronous. By default, it is asynchronous.

- **Success():**

  Takes a call back method as input, and invokes the call back method automatically when there is a response from the server.

- **Error():**

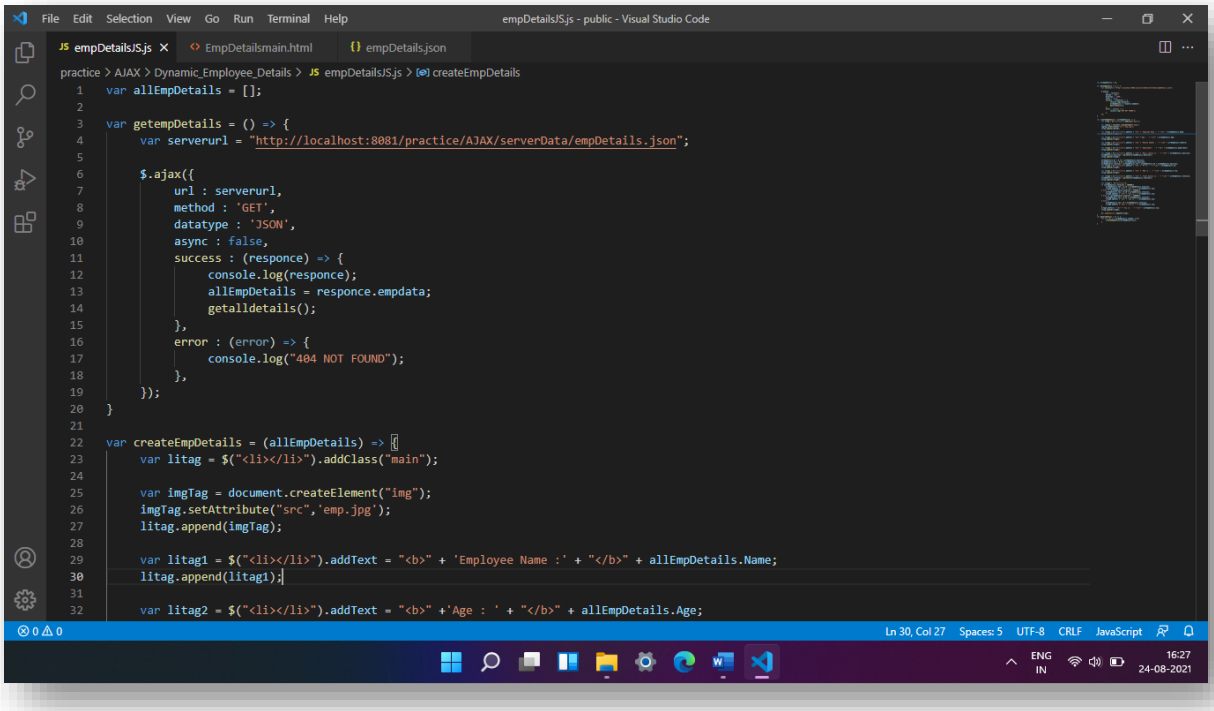
Takes a call back method as an input, and invokes the call back method automatically when there is an error while interacting with the server.

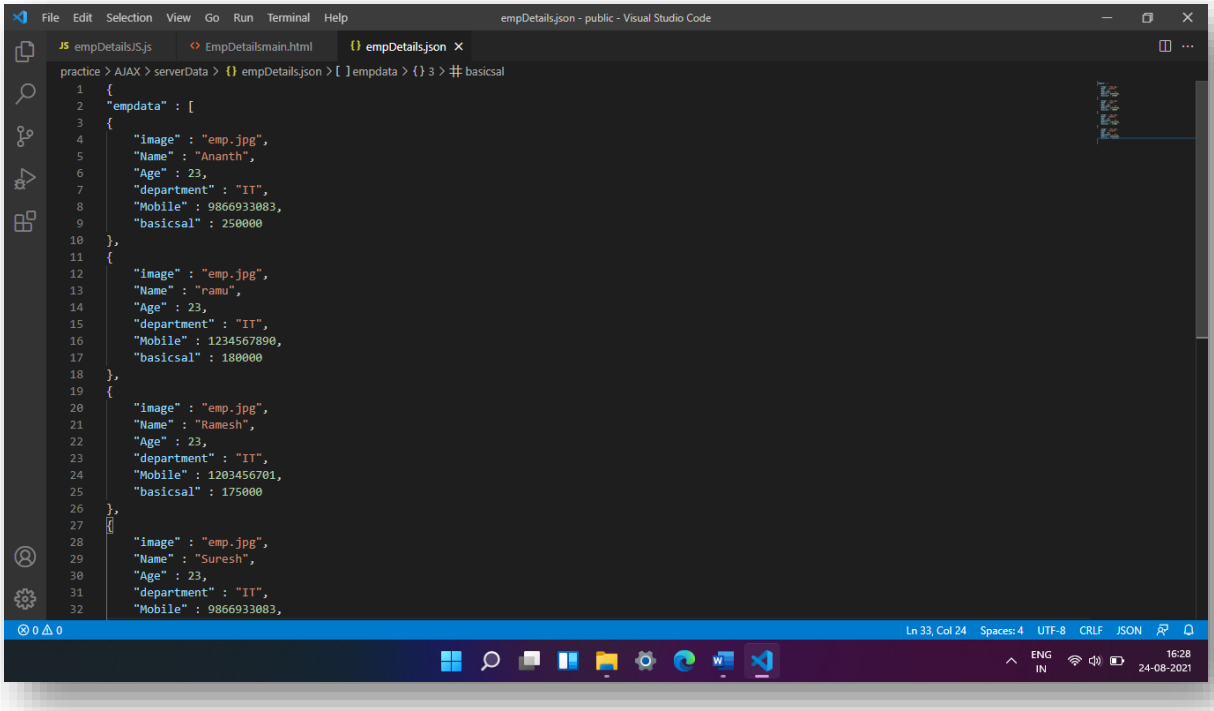
Syntax:

```
$.ajax({
        url : 'webserviceurl',
        method : 'Get/Post',
        datatype : 'Json/xml/string',
        success : function (<response>) {
                ……
                ……. // code to handle response
        }
        error : function (error) {
                …..// code to handle error
        } });
```

Ex:

JS Program:





➢ Local Storage and Session Storage:

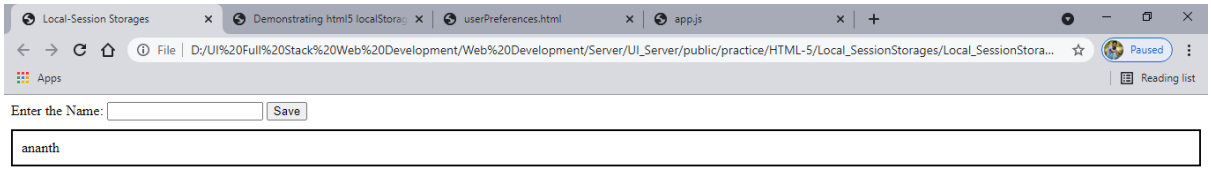
The two predefined objects supported in a **HTML-5** based browsers using which we could able to store user preferences within the browser cache itself.

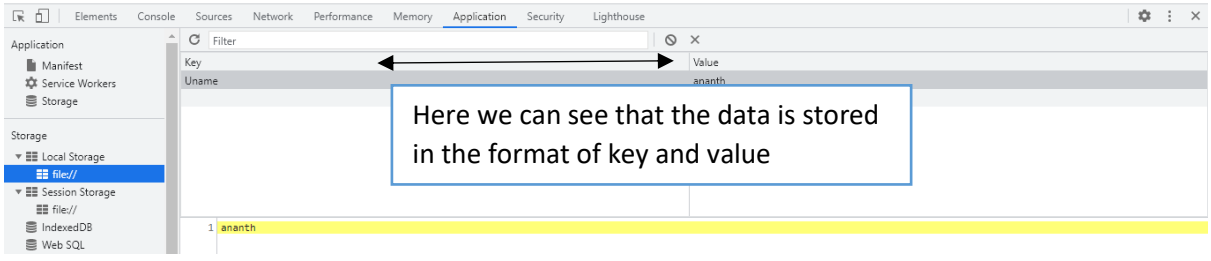
Following are the predefined methods being supported or can be applied on both the Local Storage and Session Storage object through which we could able to add, remove or update data.

- **setItem("Key",<value>):** To set a value in a cache.
- **getItem("Key"):** Returns values been stored on a key.
- **removeItem("Key"):** Removes a key value.
- **removeAll():** Removes all values inside a object.

Ex:

```
var getuname = () => {
     var name = $( "#uname" ).val();
     localStorage.setItem( "Uname", name );
     displayuname( "Uname" );
}
var displayuname = ( uname ) => {
     $( ".displayUserName" ).text( uname ).show( 3000 );
}
if ( localStorage.getItem( "Uname" ) != null ) {
     displayuname( localStorage.getItem( "Uname" ) );
}
```





The same is applicable for session storage but the only difference is

- These two objects are almost similar used to store the user preferences within the browser cache which can be assessable even after the reload or even the reopen of the browser and the only difference between these two objects is data stored under local storage object will be available even after reloading or the reopen whereas data stored under the session object will be only available on the reload of the page, and will be flushed out when we close the page.

Note: These two objects have same set of predefined methods can be applied on them.

➢ **Semantic tags of HTML-5:**

Following are the predefined semantic tags been supported in the HTML-5. The name of the semantic tags describes the purpose of the element and type of the content that is within the tag.

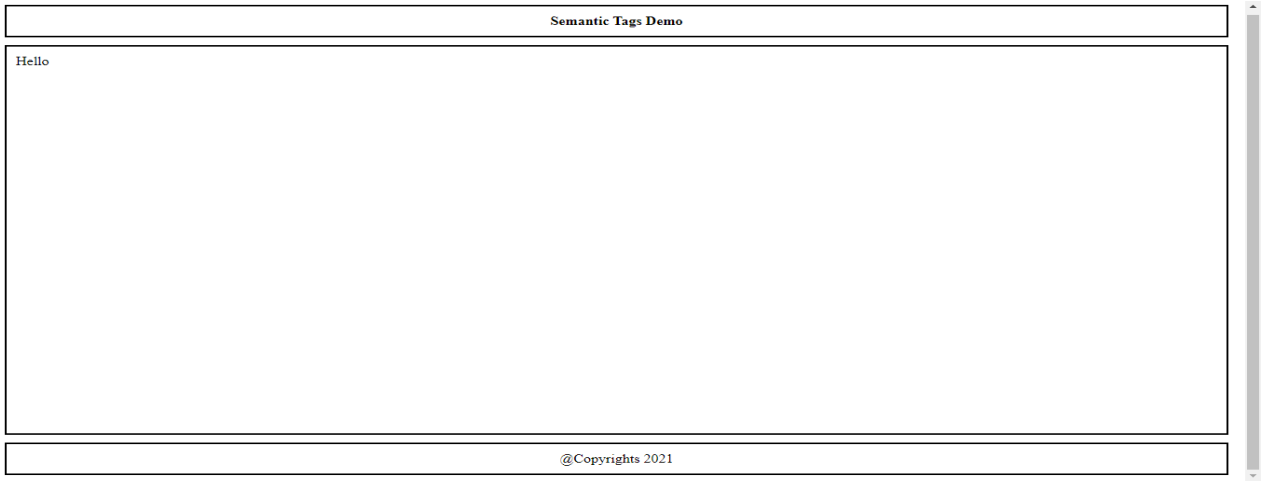
-article -aside -header -footer -maincontainer

-details -figure -main -section -navbar -nav

-summary -title etc…

Ex:

```
<body>
        <header class="head"><b>Semantic Tags Demo</b></header>
        <main class="inner">
             <selection>
                  <article>Hello</article>
             </selection>
        </main>
        <footer class="foot"> @Copyrights 2021 </footer>
   </body>
```



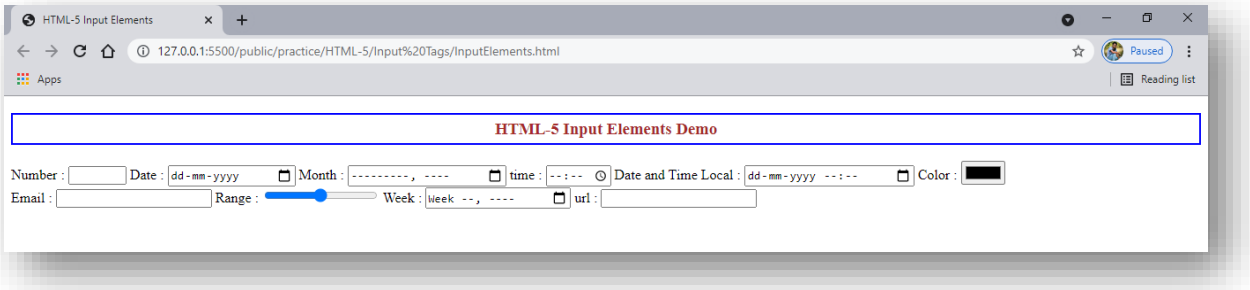
➢ HTML-5 Input Elements:

Following are the set of input elements been supported in **HTML-5** to read the different types of data from the user,

1. Input type = “color”
2. Input type = “date”
3. Input type = “datetime-local”
4. Input type = “email”
5. Input type = “search”
6. Input type = “month”
7. Input type = “number”
8. Input type = “range”
9. Input type = “tel”
10. Input type = “url”
11. Input type = “week”
12. Input type = “time” etc…

Ex:

```
<body>
        <h3>HTML-5 Input Elements Demo</h3>
        Number : <input type="number" name="Number" id="num" max="10" min="0">
        Date : <input type="date" name="date" id="dt">
        Month : <input type="month" name="month" id="mth">
        time : <input type="time" name="time" id="tm">
        Date and Time Local : <input type="datetime-local" name="dt" id="dtime">
        Color : <input type="color" name="color" id="clr"><br>
        Email : <input type="email" name="email" id="mail">
        Range : <input type="range" name="rn" id="rn" max="100" min="10">
        Week : <input type="week" name="week" id="week">
        url : <input type="url" name="ur" id="ur">
    </body>
```



➢ HTML-5 audio and video tags:

HTML-5 provides a feature of adding audio and video without any dependences on the third-party plugins.

Syntax: **<audio autoplay controls>**

<source src =”music/data/song.mp3” type = “audio/mp3”>

</audio>

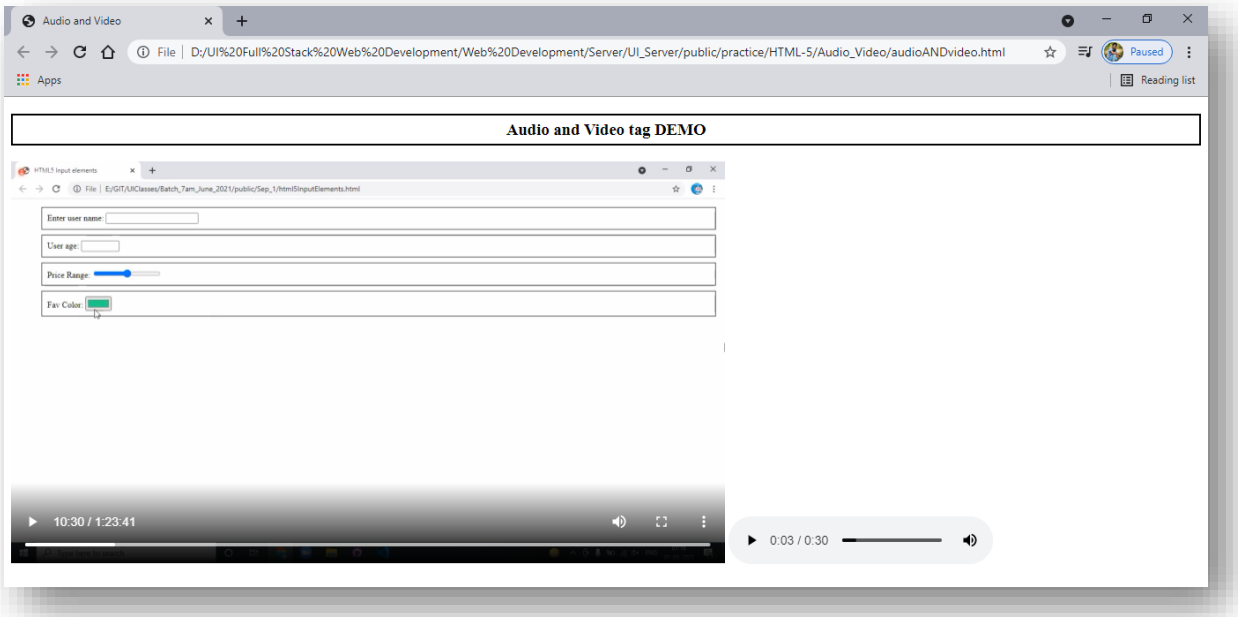
<video autoplay controls>

<source src =”video/data/vd.mp4” type = “video/mp4”>

</audio>

Ex:

```
<video controls>
            <source
                src="
D:/UI Full Stack Web Development/Web Development/Recordings/Sep/01-09-2021 (HTML_Aud_vid_navigator_appCache)/01-09-2021 (HTML_Aud_vid_navigat
or_appCache).mp4
"
                type="video/mp4" />
        </video>
        <audio controls>
            <source src="gopalagopa_RKkUDrxC.mp3" type="audio/mp3" />
        </audio>
```



- **<u>Attribute that can be added to video and audio tags:</u>**
  - **Controls:** Specifies that weather audio/video controls like play, pause etc… to show on the page.
  - **autoplay:** It says that the audio or video to be played on the load of the page itself.
  - **Height/width:** To set the height and width of the player.

- **Loop:** Boolean value specifies to keep playing audio or video file once it finishes.
- **Muted:** Specified audio/video output to be muted.
- **Poster:** Takes an image URL as input and shows the image before the video gets plays just like as thumbnail.
- **Preload:** Automatically buffers the video/audio file.

➢ **Application Cache:**

Its a new feature being supported in HTML-5 through which we could make the webpage resources to be accessible even when it is in offline.

Following are the steps to be followed to implement application cache to a webpage,

Step 1: Create an external app cache file (.appcache extension is recommended)

Step 2: Define set of rules in the appcache files which specifies which resources to be available offline, for which resources the network connection is mandatory.

Step 3: Within the HTML tag through manifest attribute specify the app cache file to be used to the current file.

```
<html Lang="en" manifest="../sample.appcache">
```

- **Creating an appcache file:**

Any appcache file contains the following three blocks,

- Cache management/ manifest
- Network
- Fallback

❋ **Cache Management:**

Under this block we specify all the resource (HTML, CSS, JS, images…) which need to be accessed even when there is network connection.

❋ **Network:**

Under this, we specify the list of resources which should never be cached, should have network connection to access these resources (Ex: LoginPage).

❋ Fallback:

Under this, we specify a custom user defined page not found file which will be automatically thrown when the user tries to access the resources which need an internet connection.

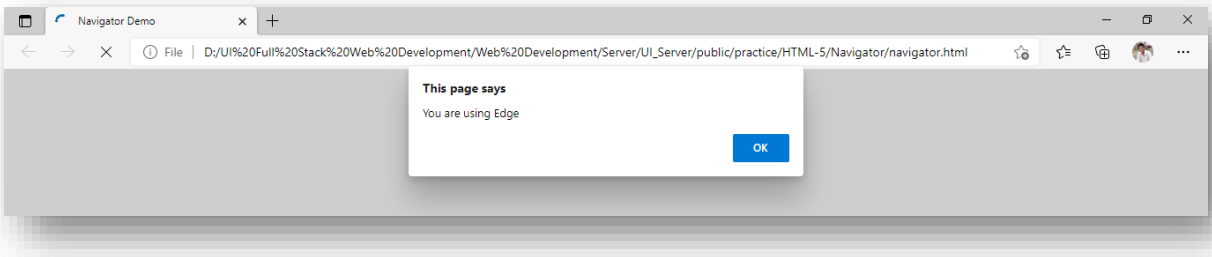
```
CACHE MANIFEST
# 2010-06-18:v3

# Explicitly cached entries
index.html
css/style.css

# offline.html will be displayed if the user is offline
FALLBACK:
/ /offline.html

# All other resources (e.g. sites) require the user to be online.
NETWORK:
*

# Additional resources to cache
CACHE:
images/logo1.png
images/logo2.png
images/logo3.png
```

➢ Navigator object:

A predefined object by default available in HTML-5 based page which holds extra information of the current browser and operating system of the current machine. It consists of,

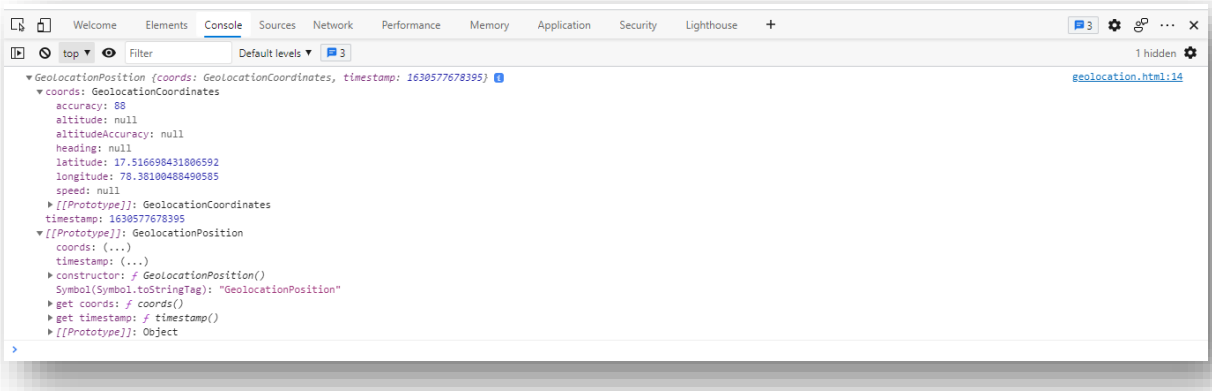
- List of languages that the browser is supporting.
- Whether the system is connected to the internet or not.
- Appcode and appname of the current browsers.
- Vendor name and version number of the current browser.
- List of plugins been installed by the browser.
- List of media devices been connected.
- Bluetooth information and geolocation information.

```
<script>
            console.log( navigator );
            if ( navigator.userAgent.includes( 'Edg' ) ) {
                alert( "You are using Edge" );
            } else if ( navigator.userAgent.includes( 'Firefox' ) ) {
                alert( "You are using firefox" );
            } else {
                alert( "You are using chrome" );
            }
        </script>
```



- **Navigator Geolocation:**

```
<body>
        <script>
            navigator.geolocation.getCurrentPosition( ( point ) => {
                console.log( point );
            } );
        </script>
    </body>
```



➢ HTML5 Canvas Tag:

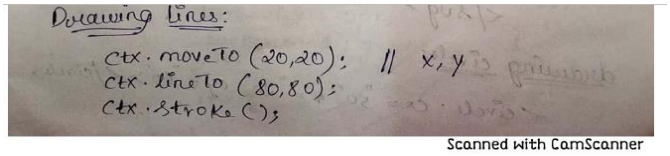
A predefined tag been supported in HTML5 using which we could able to draw the graphical objects within the canvas container.

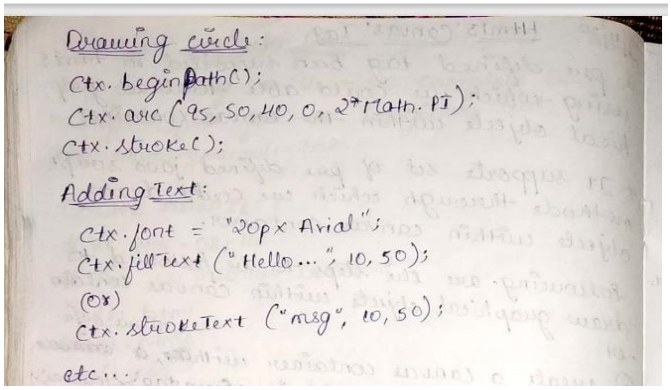
It supports set of predefined JavaScript methods through which we could draw the objects within the canvas container.

Following are the steps to be followed to draw the graphical objects within the canvas container,

- Create a canvas tag with an ID Ex: <canvas id=”usc”></canvas>
- Create a context object of canvas container.
  Ex: var element = $(“#usc”);
  //context object
  Var ctx = element.getContext(“2d”);

- Use the predefined methods on the context object through the objects can be drawn on the container.





➢ SVG Tag:

It is another way of creating graphical objects within the HTML5 Container, drawing graphical objects within the SVG is almost like a canvas tag, where the only difference is to draw objects within the canvas, we make use of JS methods where as in SVG we use predefined HTML Tags to draw objects.

Following are the predefined graphical objects can be used within the SVG container,

```
<body>
        <SVG>
            <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />

            <rect x="100" y="150" width="400" height="100"
                style="fill:rgb(0,0,255);stroke-width:10;stroke:rgb(0,0,0)" />

            <ellipse cx="200" cy="80" rx="100" ry="150" style="fill:yellow;stroke:purple;stroke-width:2" />

            <line x1="0" y1="0" x2="200" y2="200" style="stroke:rgb(255,0,0);stroke-width:2" />

            <text x="0" y="15" fill="red">Good morning all</text>
        </SVG>
    </body>
```

➢ HTML Drag and Drop Events:

HTML5 supports handling new set of event types like drag and drop. We could able to invoke call back methods when an element is getting dragged or dropped.

- Draggable events are **dragstart**, **dragend** and attribute is **draggable = "true".**
- Drop target events are **dragenter, dragover, dragleave, drop.**

```
<body>
        <div class="container" ondrop="elementDropped(event)" ondragover="dagIsOver(event)"></div>
        <img id="prod_1" ondragstart="draggingIsStarted(event)"
            src="https://m.media-amazon.com/images/I/71h6PpGaz9L._AC_SL1500_.jpg" alt="" />
        <img id="prod_2" ondragstart="draggingIsStarted(event)"
            src="https://bazarexonline.com/crm/public/uploads/subcategory_images/9msLDJjDS1Gun9ZrAQl1Onx3Dikg6K.jpg"
            alt="" />
        <img id="prod_3" ondragstart="draggingIsStarted(event)"
            src="
https://www.cnet.com/a/img/xh7SVje9bq1_azlPW73EGrHHJVQ=/1200x630/2020/10/28/5a601f4a-06c8-4374-b10d-9f07fe183390/babyyoda-target-1.png"
            alt="">

        <script>
            var productId = '';
            var elementDropped = ( event ) => {
                event.preventDefault();
                productId = '#' + productId;
                event.target.append( document.querySelector( productId ) );
            }

            var dagIsOver = ( event ) => {
                event.preventDefault();
            }

            var draggingIsStarted = ( event ) => {
                console.log( "drag is started" );
                productId = event.target.getAttribute( "id" );

            }
        </script>
    </body>
```

- **<u>Multitasking and multithreading:</u>**

The concept of process of executing multiple jobs at a time to increase the performance of the page is called as multitasking or multithreading.

Note: JavaScript does not support multitasking directly but we can make use of it indirectly.

➢ <u>HTML5 Web Worker:</u>

It is a new feature been supported in HTML5 through which we can indirectly achieve the support of multitasking and multithreading,

- Workers are a separate JS file gets initiated through the main thread, executes parallelly to the main thread.
- Even through the web worker gets executed independent through the main thread it can still pass or communicate to the main thread.
- Even the main thread is capable of receiving messages from a worker.
- In a single page we can use any number of web workers.
- Following are the steps to be followed to implement web workers in an application,

Step-1: Create an external JS file (web worker is a JS file) which executes independent to the main thread using the POST message method.

Step-2: Instantiate a web worker from the main thread through the predefined worker class.

Syntax: var worker = new Worker ("Worker JS File path");

Step-3: Add the onmessage event handler on worker object which gets involved automatically when there is a response from corresponding web worker.

Syntax: worker.onMessage = function(event) {

….// call back method gets fixed automatically when there is a message from the web worker.

}

Ex:

```
var count = 1;
var doJob = () => {

     var myWorker = new Worker( 'dateWorker.js' );
     myWorker.onmessage = function ( event ) {
          document.querySelector( ".dateContainer span" ).innerHTML = event.data;
          // myWorker.terminate();
     }

     // Creating Container.
     setInterval( () => {
          var divTag = document.createElement( "div" );
          divTag.setAttribute( "class", 'content' )
          divTag.innerHTML = "Content Created " + count;
          count++;
          document.querySelector( ".main-container" ).append( divTag );
     }, 1000 );
}
```

```
var temp = 0
function getDateInFormat () {
     setInterval( function () {
          var date = new Date();
          var reqFormat = date.getMonth() + '/' + date.getDate() + '/' + date.getFullYear() + ' ' + date.getHours() + ':' + date.getMinutes
() + ':' + date.getSeconds();
          console.log( "Date format is " + reqFormat );
          temp++;
          //var data = {msg: 'one', data: 'twe'};
          //postMessage(JSON.stringify(data));
          postMessage( reqFormat );
     }, 1000 );

}


getDateInFormat();
```

CSS-3

➢ Border-Radius:

The **border-radius** CSS property rounds the corners of an element's outer border edge. You can set a single radius to make circular corners, or two radii to make elliptical corners.

Ex: border-radius: 30px;

border-radius: 25% 10%;

border-radius: 10% 30% 50% 70%;

- The radius applies to the whole background, even if the element has no border; the exact position of the clipping is defined by the background-clip property.
- The border-radius property does not apply to table elements when border-collapse is collapse.

➢ Constituent properties:

This property is a shorthand for the following CSS properties:

- border-top-left-radius
- border-top-right-radius
- border-bottom-right-radius
- border-bottom-left-radius

The border-radius property is specified as:

- one, two, three, or four <length> or <percentage> values. This is used to set a single radius for the corners.
- followed optionally by "/" and one, two, three, or four <length> or <percentage> values. This is used to set an additional radius, so you can have elliptical corners.

<length>:

Denotes the size of the circle radius, or the semi-major and semi-minor axes of the ellipse, using length values. Negative values are invalid.

<percentage>:

Denotes the size of the circle radius, or the semi-major and semi-minor axes of the ellipse, using percentage values. Percentages for the

horizontal axis refer to the width of the box; percentages for the vertical axis refer to the height of the box. Negative values are invalid.

- **Multibackground Images:**

You can apply **multiple backgrounds** to elements. These are layered atop one another with the first background you provide on top and the last background listed in the back. Only the last background can include a background color.

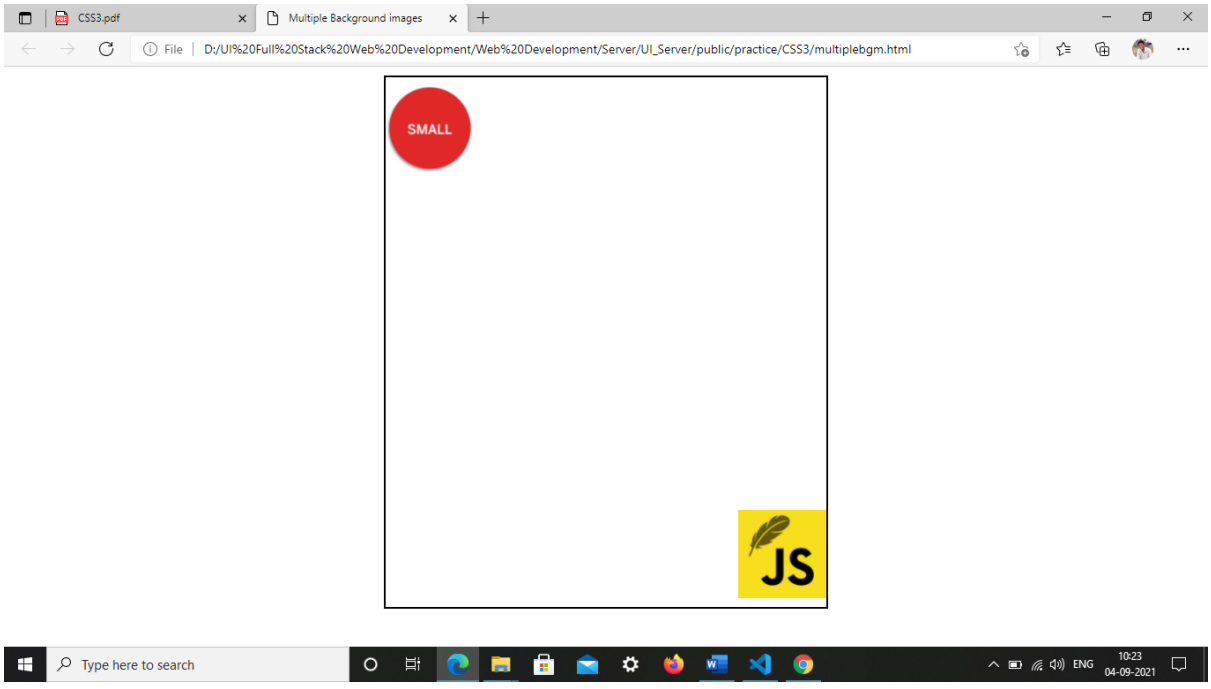
Specifying multiple backgrounds is easy:

Background-image : url(“abc.jpg”),url(“abcd.png”);

Background-position : left top, right bottom;

Ex:

```
.containers {
                    border: 2px solid;
                    margin: 10px auto;
                    width: 500px;
                    height: 600px;
                    background-image: url("
https://images.prismic.io/smallexchange-com/21260cc6-c545-4571-8f59-cd96052b327e_SMFE_Site_Buttons-SMALL@3x.png?auto=compress,format    "),
url("https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSpMHRgGhGu9aTtu0Te1mcG91JjK6v241Hbbw&usqp=CAU");
                    background-repeat: no-repeat;
                    background-size: 100px;
                    background-position: left top 10px, right bottom 10px;
```



➢ CSS Box-Shadow:

The box-shadow CSS property adds shadow effects around an element's frame. You can set multiple effects separated by commas. A box shadow is described by X and Y offsets relative to the element, blur and spread radius, and color.

Ex: box-shadow: 1px 2px 5px 5px red;

box-shadow: 12px 12px 2px 1px rgba(0, 0, 255, .2);

Ex:

```
 /* Keyword values */
box-shadow: none;

/* offset-x | offset-y | color */
box-shadow: 60px -16px teal;

/* offset-x | offset-y | blur-radius | color */
box-shadow: 10px 5px 5px black;

/* offset-x | offset-y | blur-radius | spread-radius | color */
box-shadow: 2px 2px 2px 1px rgba(0, 0, 0, 0.2);

 /* inset | offset-x | offset-y | color */
box-shadow: inset 5em 1em gold;

/* Any number of shadows, separated by commas */
box-shadow: 3px 3px red, -1em 0 0.4em olive;

              /* Global keywords */
              box-shadow: inherit;
              box-shadow: initial;
              box-shadow: revert;
              box-shadow: unset;
```

➢ CSS3 transitions and transformation:

Transitions and transformations are the new set of CSS properties through which we could able to control the transformation of the DOM Element while rendering, adding the animation delay through transition.

✵ CSS Transform property:

Any DOM Element can be transformed in following three different ways,

- Rotate - translate - scale - skew

- **Transform with Rotate property:**

  Through which any DOM Element can be rotated within 0° to 360°

  Ex: transform : rotate(90deg);

  transform : rotate(x/y)(90deg);

```
    <style>
        .cont {
            border: 2px solid;
            width: 200px;
            height: 120px;
            margin: 10px auto;
            /*transform: rotate(90deg);*/
        }

    </style>
</head>

<body>
    <div class="cont"></div>
    <script>
        var degree = 10;
        setInterval( () => {
            document.querySelector( ".cont" ).style.transform = "rotate" + "(" + degree + "deg" + ")";
            degree += 10;
        }, 0.1 )
    </script>
```

- **Transform with translate property:**

  Through which we could able to move DOM Element to a new position with provided values.

  Ex: transform: translate(30px, 20px);

```
<style>
        .cont {
            border: 2px solid;
            border-radius: 100px;
            width: 200px;
            height: 120px;
            position: absolute;
            top: 100px;
            left: 200px;
            box-shadow: 1px 2px 5px 2px;
        }

        .cont:hover {
            transform: translate(100px, 200px);
        }

    </style>
```

- **Transform with Scale:**

  Through which we can redraw the elements with effects,

  Ex: transform: scale(3,4) ;

  // redraws the DOM element with 3times of width and 4 times of height

```
<style>
            .cont {
                border: 2px solid;
                border-radius: 100px;
                width: 200px;
                height: 120px;
                position: absolute;
                top: 300px;
                left: 200px;
                box-shadow: 1px 2px 5px 2px;
                text-align: center;
            }

            .cont:hover {
                transform: scale(2, 4);
            }

        </style>
```

- **Transform with Scale:**

  The **skew ()** CSS function defines a transformation that skews an element on the 2D plane.

  Ex: transform: skew (15deg, 15deg);

  transform: skew (0);

```
<style>
            .cont {
                border: 2px solid;
                border-radius: 100px;
                width: 200px;
                height: 120px;
                position: absolute;
                top: 300px;
                left: 200px;
                box-shadow: 1px 2px 5px 2px;
                text-align: center;
            }

            .cont:hover {
                transform: skew(2deg, 14deg);
            }

        </style>
```

- The skew () function is specified with either one or two values, which represent the amount of skewing to be applied in each direction. If you only specify one value it is used for the x-axis and there will be no skewing on the y-axis.

  Syntax: skew(ax);

  //ax is an angle representing the angle to use to distort the element along the x-axis.

  skew (ax, ay);

  // ax is an angle representing the angle to use to distort the element along the x-axis. ay is an angle representing the angle to use to distort the element along the y-axis.

➢ **CSS Keyframes:**

The **'@keyframes'** or the set of CSS rules which lets the user to control the intermediate steps in CSS animation sequence by establishing keyframes along with the animation sequence which must be reached to a certain point during the animation.

- In order to make use of keyframes we create a keyframe rule with the user defined name, we use CSS animation property to match its animation to its keyframe list.
- Each keyframe rule contains a style list a keyframe selectors each of which contains a percentage along with the animation at which the key frame occurs as well as a block containing transformation for that key frame.

Syntax:

```
@keyframes slidein {
  Value1 % {
    transform: translateX(0%);
  }

  Value % {
    transform: translateX(100%);
  }
}
```

Ex:

```
<style>
        @keyframes moving {
            10% {
                background-color: red;
                top: 200px;
                transform: rotate(360deg);
            }

            40% {
                background-color: blue;
                top: 100px;
                left: 300px;
            }

            100% {
                background-color: aqua;
                bottom: 100px;
                transform: rotate(360deg);
            }
        }

        .cont {
            border: 2px solid;
            width: 100px;
            height: 180px;
            position: absolute;
            animation: moving 5s infinite;
        }

    </style>
```

- **CSS Linear Gradient:**

The **linear-gradient ( )** is a CSS function which creates an image consisting of a progressive transition between two or more colors along a straight line.

Using CSS3 **linear-gradient ( )** we could add smooth transitions two or more specified colors.

Syntax:

background-image: linear-gradient (to <direction>, color1, color2);

Direction could be,

to left, to right, to left bottom, to right bottom etc…

- **With direction as angle:**

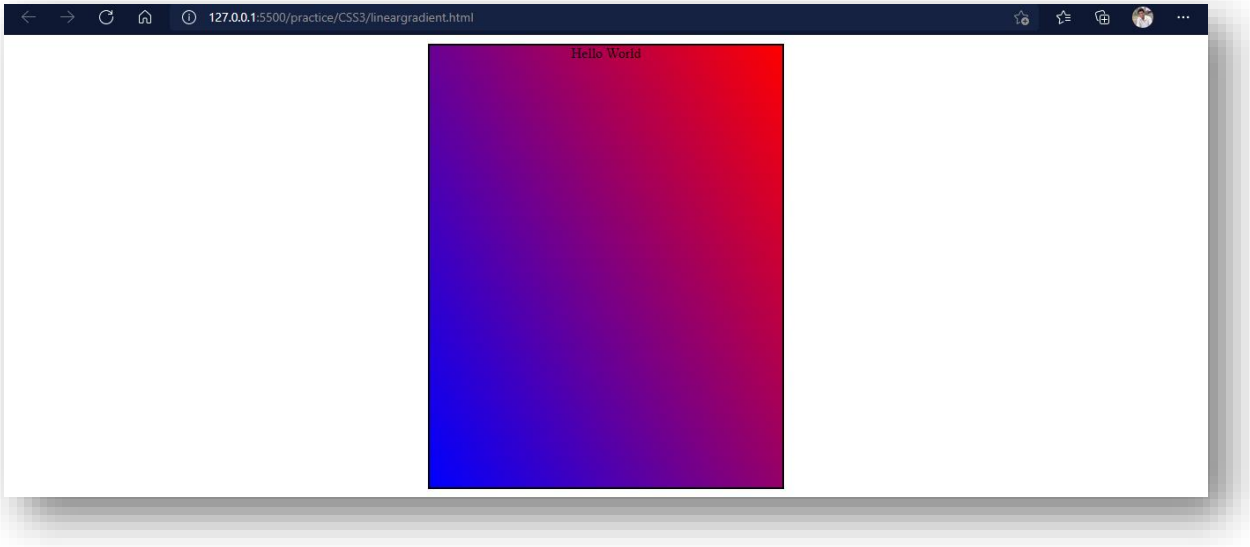
background-image: linear-gradient (<angle>, color1, color2);

- **Repeat linear-gradient():**

background-image: repeating-linear-gradient (to <direction>/angle, color1, color2);

Ex:

```
.inner {
                border: 2px solid;
                width: 400px;
                height: 500px;
                margin: 10px auto;
                text-align: center;
                background-image: linear-gradient(240deg, red, blue);
            }
```



- **Radial-gradients( ):**

Using which the gradients can be specified with in the centre.

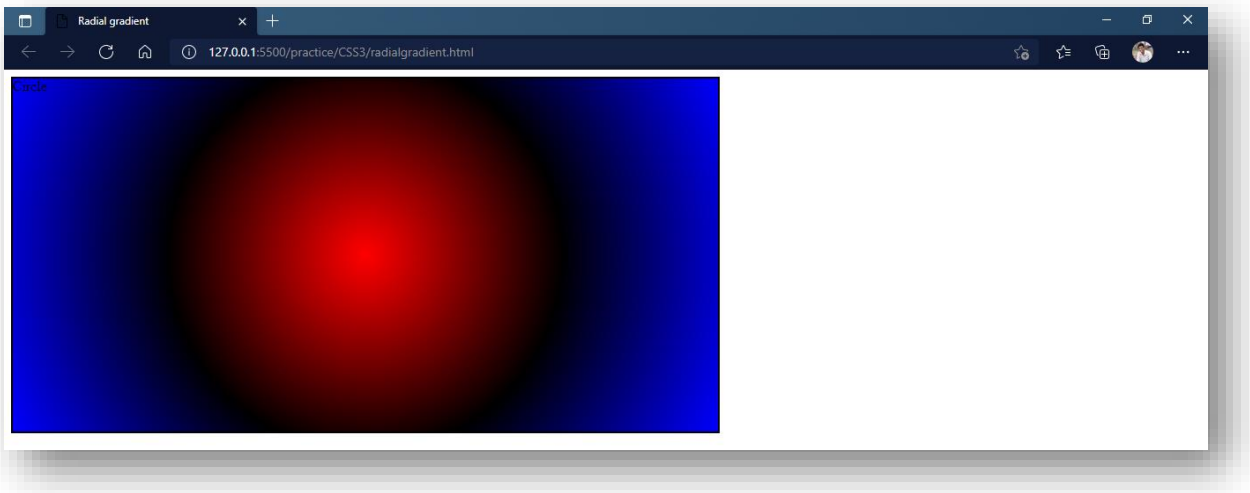
By default, the **Radial-gradients( )** shape is ellipse and position is enter, we can still change the position and shape.

background-image: radial-gradient (color1, color2,…);

background-image:radial-gradient (circle, color1 5%, color2 10%,…);

Ex:

```
.rad {
                border: 2px solid;
                width: 800px;
                height: 400px;
                background-image: radial-gradient(circle, red, black, blue);
            }
```



➢ CSS3 Media Queries:

A predefined feature being supported in CSS3 through which we could able to design multiple set of CSS Classes for different device dimensions so that corresponding CSS will be applied for particular device dimensions.

"@media" is a keyword through which we define set of CSS Classes for particular dimensions.

Syntax: @media <mediatype> and (dimension) {

…..

}

Following are the set of properties through which we can specify device dimensions within media queries,

- Min-width
- Min-height
- Max-height
- Max-width

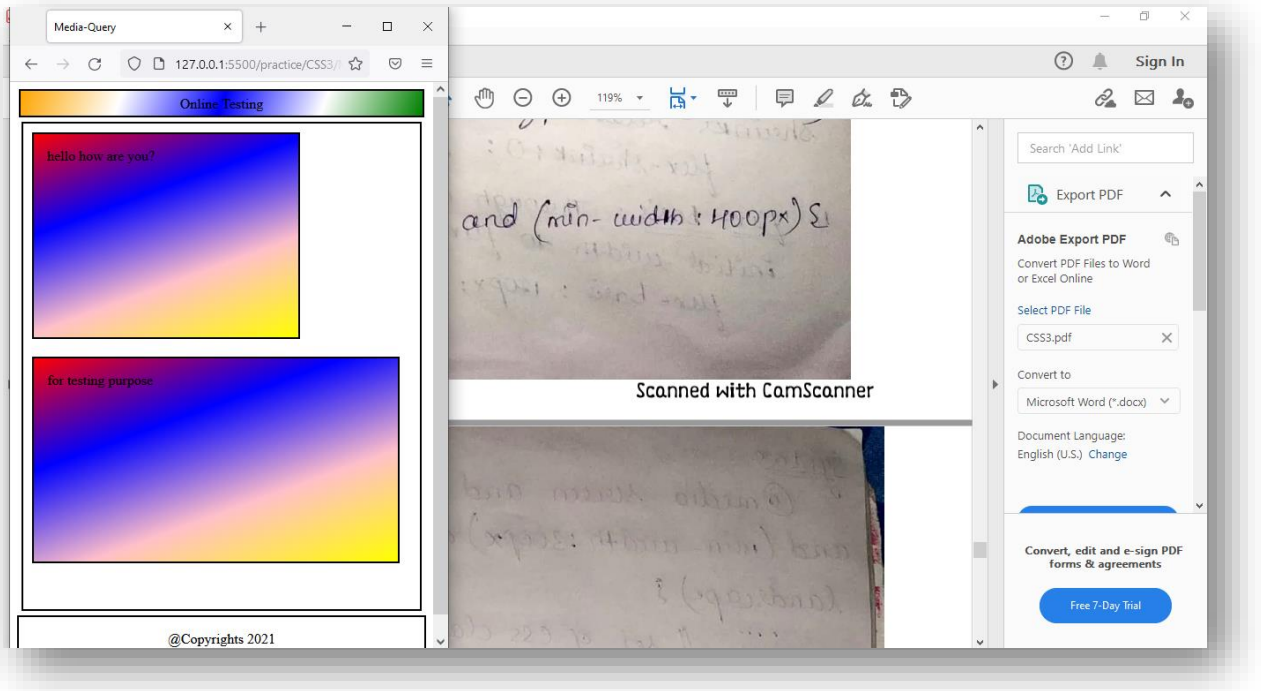
Following are the types of media can be specified while defining media queries,

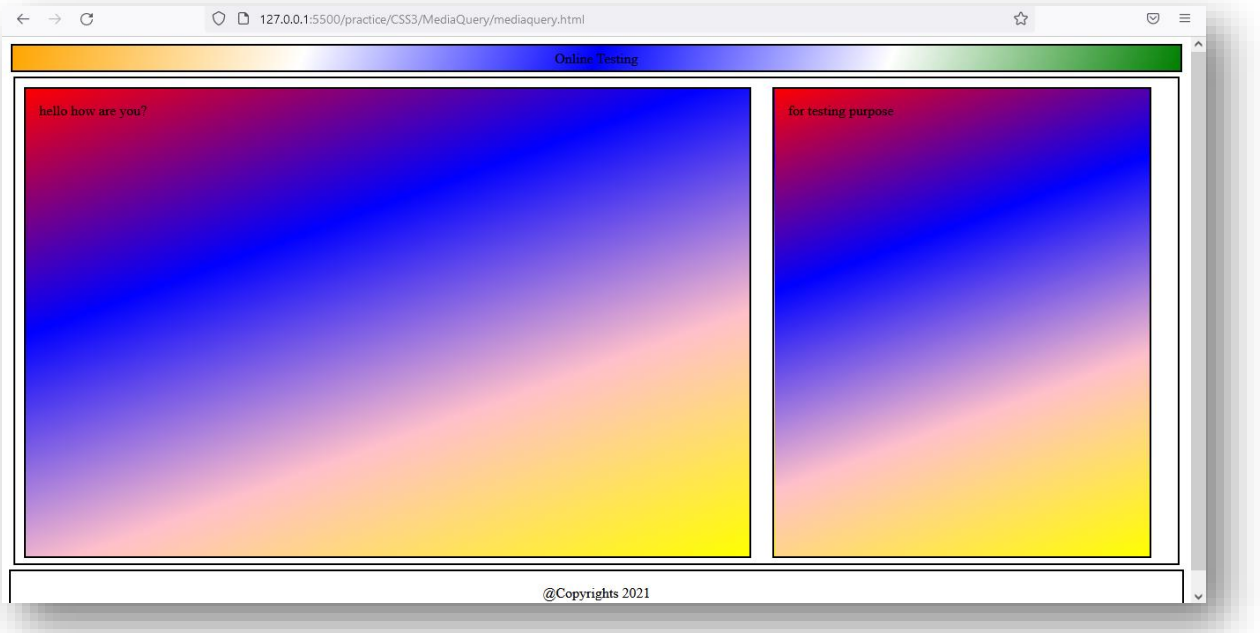
Media Types:

- **Screen –** Any color screen
- **All –** indicates all screens
- **Handheld –** indicates all small screen projections
- **Print –** indicates print devices.

Ex:

```
@media screen and (min-width : 480px) and (max-width : 585px) {
     .inner1 {
          height: 200px;
     }
     .inner2 {
          height:200px;
          width : 85%;
     }
}
```





- While defining the media-queries we can specify the device orientation using orientation property.
- **"orientation"** possible values **"Landscape/portrait".**
- **Syntax: @media Screen and (max-width:300px) and (min-width:200px) and (orientation : landscape/portrait) {**

  …….. //set of CSS Classes

  }

- The media queries can be injected into the website by **@media queries** or **<link rel="stylesheet" media="max-width:200px; min-height:400px"; orientation : portrait" href="data/app.css">**

➢ **CSS Flex:**

Following are the five layout models been supported in CSS,

- Block layout
- Inline layout
- Table layout
- Positioned layout
- Flex box layout

➢ **Flex Box Layout:**

Flex container becomes flexible by setting the display property to flex. **"display:flex;"** through Flex Box Layout module we could design responsive layout structures without any dependency of either float or position.

Following are the properties can be applied to flex container.

- Flex-direction
- Flex-wrap
- Flex-flow
- Justify-content
- Align-items
- Align-content

➢ **Flex direction property:**

Through which we can specify in which the direction of the flex items to be rendered.

Following are the possible values it takes,

- Column
- Row

- Column-reverse
- Row-reverse

➢ **Flex-Wrap:**

Through which the items with flex container gets wrapped if needed.

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

➢ **Flex-flow:**

It's a shortcut way to define both direction and wrap property.

Flex-flow : row wrap;

➢ **Justify-content:**

Through which we can align the items under flex container.

justify-content : center / flex-start / flex-end / stretch / baseline.

➢ **Align-items:**

Through which we could align flex items.

Align-items : center / flex-start / flex-end / stretch / baseline.

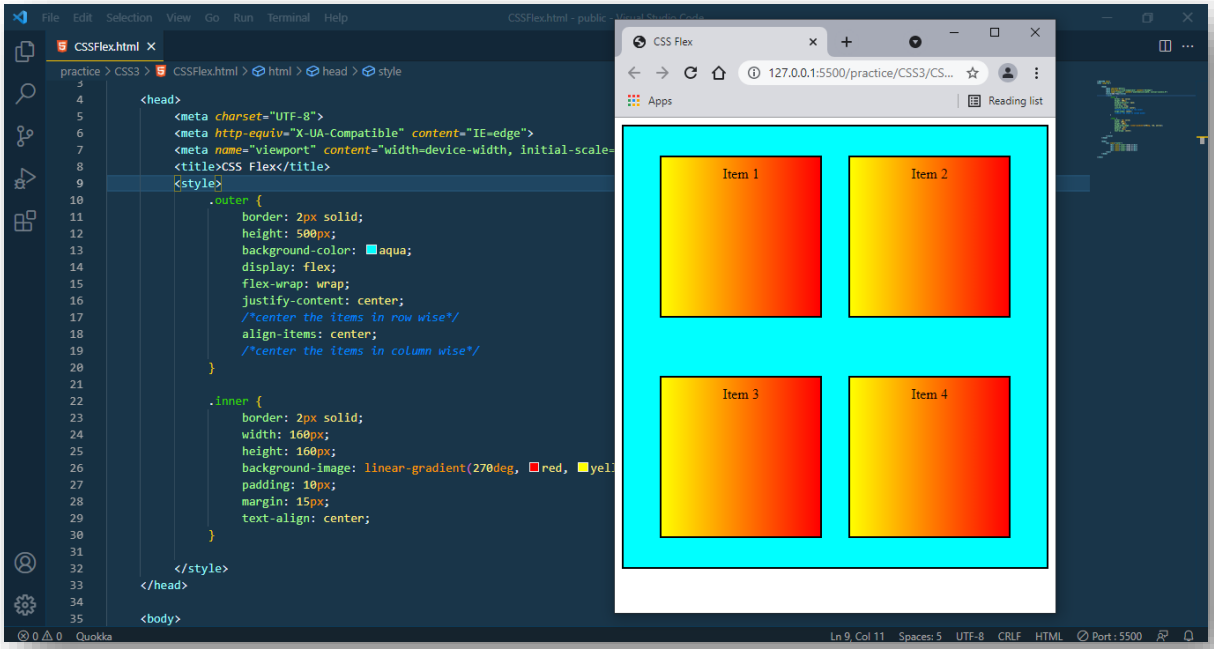
➢ **Align-content:**

To align content inside flex container.

flex-wrap : wrap;

align-content : space-between / space-around / stretch / center;

```
<style>
            .outer {
                border: 2px solid;
                height: 500px;
                background-color: aqua;
                display: flex;
                flex-wrap: wrap;
                justify-content: center;
                /*center the items in row wise*/
                align-items: center;
                /*center the items in column wise*/
            }

            .inner {
                border: 2px solid;
                width: 160px;
                height: 160px;
                background-image: linear-gradient(270deg, red, yellow);
                padding: 10px;
                margin: 15px;
                text-align: center;
            }

        </style>
```



➢ Flex-item Properties:

Following are the properties can be applied to flex items,

- Order
- Flex-grow
- Flex-shrink
- Flex-basis
- Flex
- Align-self

➢ Order:

Through which we can control rendering order of flex item,

Order: 1/3/5…;

➢ Flex-grow:

Through which we can specify how much an item grows relative to other flex items in it,

Flex-grow: 1/3/8…;

➢ Flex-shrink:

Specifies how much item shrinks relative to other items,

Flex-shrink: 0; //no Shrink

- **Flex-basis:**

  Through which we can specify initial width to the flex item.

  Flex-basis: 120px;

- **Flex:**

  Through which we can control flex-grow, shrink and basis at a time.

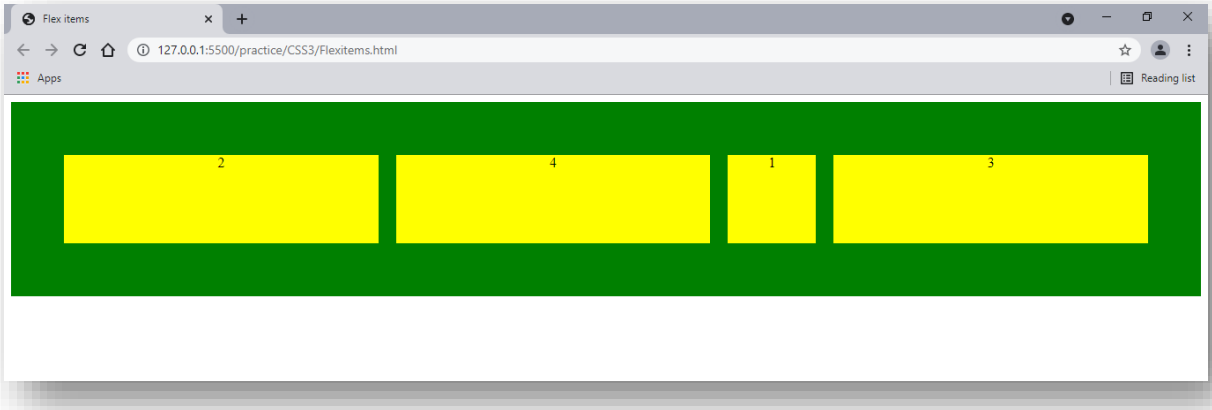
  Flex: 1 0 200px;

- **Align-self:**

  Through which we can make item to get aligned automatically within the container.

  Align-self: center;

```
<style>
            .container {
                background-color: green;
                padding: 50px;
                display: flex;
            }

            .item {
                background-color: yellow;
                width: 100px;
                height: 100px;
                text-align: center;
                margin: 10px;
            }

        </style>
    </head>

    <body>
        <div class="container">
            <div class="item" style="order:3;flex-shrink: 3; flex-wrap: nowrap;">1</div>
            <div class="item" style="order:1 ;flex-grow:1">2</div>
            <div class="item" style="order:4;flex-grow:1">3</div>
            <div class="item" style="order:2;flex-grow:1">4</div>
        </div>
    </body>
```



➢ Display Grid:

CSS Grid Layout excels at dividing a page into major regions or defining the relationship in terms of size, position, and layer, between parts of a control built from HTML primitives.

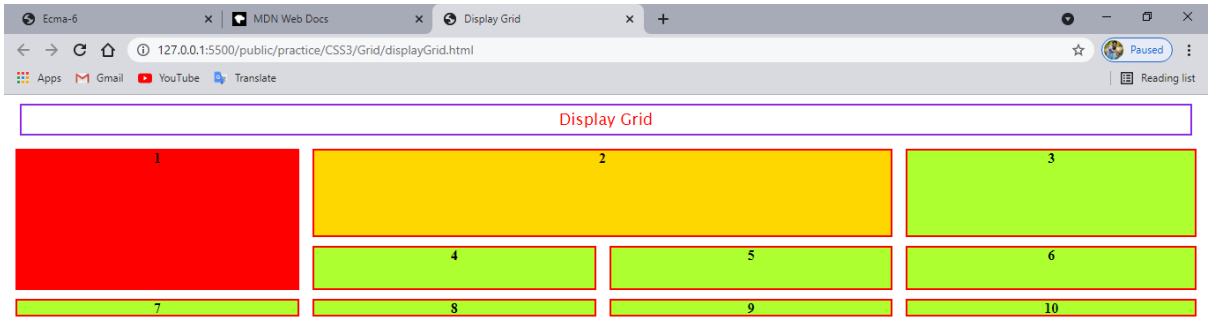
Like tables, grid layout enables an author to align elements into columns and rows. However, many more layouts are either possible or easier with CSS grid than they were with tables. For example, a grid container's child elements could position themselves so they actually overlap and layer, similar to CSS positioned elements.

Ex:

```
<style>
        header {
            border: 2px solid;
            font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode', Geneva, Verdana, sans-serif;
            font-size: large;
            text-align: center;
            border-color: blueviolet;
            color: red;
            padding: 5px;
            margin: 10px;
        }

        .container {
            display: grid;
            /* we can make grid or inline-grid */
            padding: 5px;
            grid-template-columns: auto auto auto auto;
            grid-template-rows: 100px 50px 20px;
            /*grid-column-gap: 10px;
            grid-row-gap: 10px;*/
            grid-gap: 10px 15px;
            /*10px for row wise and 15px is col wise*/
        }

        .container>div {
            border: 2px solid red;
            font-weight: bolder;
            text-align: center;
            background-color: greenyellow;
        }

        .block2 {
            /*grid-column-start: 2;
            //Starting cell
            grid-column-end: 4;
            //Before what cell it should be end*/
            grid-column: 2/4;
            background-color: gold !important;
        }

        .block1 {
            /*grid-row-start: 1;
            grid-row-end: 3;*/
            grid-row: 1/3;
            background-color: red !important;
        }

    </style>
```

HTML:

```
<body>
        <header>Display Grid</header>
        <section>
            <div class="container">
                <div class="block1">1</div>
                <div class="block2">2</div>
                <div>3</div>
                <div>4</div>
                <div>5</div>
                <div>6</div>
                <div>7</div>
                <div>8</div>
                <div>9</div>
                <div>10</div>
            </div>
        </section>
    </body>
```



BootStrap-V5.1.1

- https://getbootstrap.com/ **Get full document here.**

ECMA-6

ECMA-6:

- It is trade mark scripting language specifications standardize by **ECMA (European Computer Manufactures Association Script)** international org.
- It was created to standardize JavaScript as to faster multiple implementations.
- **ECMA-6** is also called as **ECMA 2019** which was released after a long gap after 2009.
- This release added a significant new syntax for writing complex applications including classes and modules etc…
- The other new feature includes iterators, forEach, forOf looping control structures arrow functions, type array, collections, promises etc…

Difference between let, const and var key-words:

- **var Key-word:**

  Variable gets created using **var** keyword becomes module scope.

```
Ex:   var a = 10;
      var data = ( ) => {
            var b = 20;
            res = a + b;  // here a can be accessible
            if (a > 5) {
                  var d = 10;
                  console.log(b);
                  console.log(a);      //here both a and b is accessible
            }
            Console.log(d)      // here d is accessible
      }
      Console.log(b);     // here b is not accessible outside the module.
```

- **Let Key-word:**

  Using **let** key-word we could able to create block scoped variables. The variables being created using let keyword can be accessible under a particular block and cannot be accessed outside of the block.

```
Ex:   var a = 10;
var data = ( ) => {
        let b = 20;
        res = a + b;  // here a can be accessible
        if (a > 5) {
                let d = 10;
                console.log(b);
                console.log(a);      //here both a and b are accessible
        }
        Console.log(d)      // here d is not accessible because it is
                               inside the block
}
Console.log(b);     // here b is not accessible outside the module.
```

- **Const Key-Word:**

  Variable been declared using **CONST,** once we initialize the variable we couldn't able to change the value in it.

```
Ex:   const age = 20;
        age++;  //here age cant be incremented because the value
        we declared const age = 20 is final.
```

➢ **<u>forEach looping Control Structure:</u>**

A predefined looping control structure we could able to iterate over an array which takes a call back method and throws the corresponding value and index value.

```
Syntax: a = [10,20,30];
        a.forEach ( (value,index) => { console.log(value); };
```

Ex:

```
<script>
            var a = [ 20, 30, 40 ];
            a.forEach( ( value, index ) => {
                value += 5;
                console.log( `${ index } : ${ value }` );
            } );
        </script>
```

- **JavaScript Arrow Function:**

From **ECMA-6** to define a function within JavaScript we won't be using the function keyword but we can declare the function just by using arrow symbols **(=>).**

Syntax: var sample = ( ) => {

…..

}

The main intention of using arrow functions is it holds current **this** operator. Even that particular function gets called after some time.

- **Function parameters with default values:**

From **ECMA-6** JavaScript supports a feature of having default values for formal parameters which will be considered automatically if there corresponding values were not being passed for actual parameters.

Syntax: var a = (param1 = val1, param2 = val2,…){

….

}

a(val1) //here value 2 is taken from default parameter.

Ex:

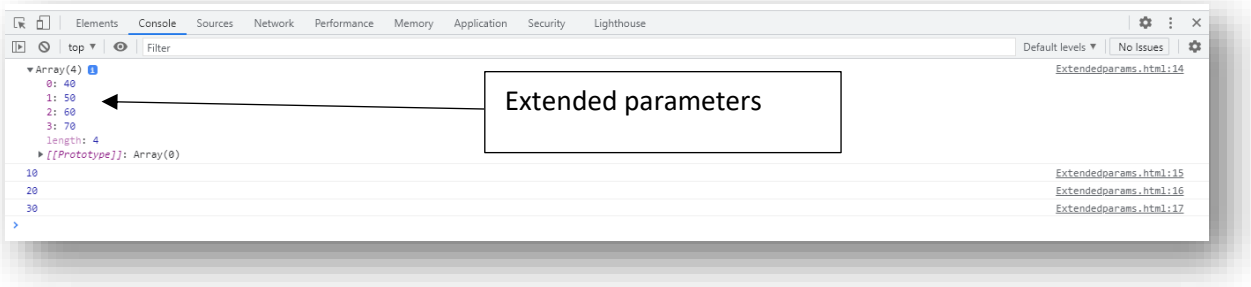
```
var param1 = ( a = 10, b = 20, c = 30 ) => {
                console.log( a );
                console.log( b );
                console.log( c );
            }
            param1( 30 );
// here a value 10 will be replaced with 30 and b and c is taken from default parameter
```

➢ Extended parameter handling:

"…" is an extended operator being supported in JavaScript using which all the parameters which are not been handled will be automatically catched in the form of an array.

Ex:

```
<script>
        var data = ( a, b, c = 10, ...e ) => {
            console.log( e );
            console.log( a );
            console.log( b );
            console.log( c );
        }
        data( 10, 20, 30, 40, 50, 60, 70 );
    </script>
```



➢ Template literals or Template strings:

Using template literals we could able to replace part of a string with given value,

Ex: var Name = 'Raj';

var Age = 60;

var msg = `username is ${Name} and age is ${Age}`;

➢ Java Script Classes:

- From **ECMA-6** JavaScript supports a keyword called class through which we could able to create a class in JavaScript.
- Classes are used to create the basic structure of an object or skeleton of an object.
- For a single class we can create any number of objects.
- **"new"** is a predefined keyword through which we could able to instantiate a class in JavaScript.
- **"Constructor"** is an predefined keyword through which we could able to add a Constructor in JavaScript.

- **"Super"** is a predefined keyword through which we can invoke constructor of a parent class through child class.
- In JavaScript we cannot override constructor method.

Ex:

```
<script>
        class Parent {
            constructor() {
                this.name = 'Ananth';
                this.age = 20;
            }
        }
        class Child extends Parent {
            constructor() {
                super();
                this.school = 'GMS';
            }
            print() {
                console.log( `Name is ${ this.name } and age is ${ this.age } and school is ${
this.school }` );
            }
        }

        var obj1 = new Parent();
        var obj2 = new Child();
        obj2.print();
    </script>
```

➢ **Exponentiation operator:**

""** is a new exponentiation operator supported from ECMA-6 through which we could able to find a power of numbers.

Ex: let a = 23;**

➢ **Array assigned values:**

If multiple values need to be assigned at a time, we could able to declare the values under an array and assign array of values to it.

Ex: [a, b, c] = [10, 20, 30];

➢ **Set's and Map's Data Structure:**

The two predefined data structures been supported in ECMA-6,

➢ **Set data Structure:**

It is almost like an array data structure used to hold group of relative items to identify the content with same variable name. The only difference

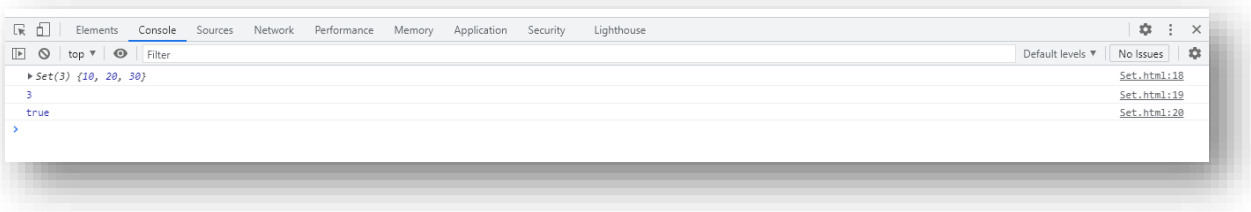
between sets and arrays are set accepts only unique values, doesn't allow duplicate values.

Syntax: var demo = new Set( ); //Creates a Set DataStructure.

- **Following are some predefined methods that can be applied on SET:**
  - **add(<value>) // adds values to Set**
  - **set //returns total number of values in set**
  - **values( ) //returns values with set**
  - **has("value")//returns a Boolean value in provided string present set**
  - **clear //clears all the values from set.**
  - **Delete("value") //Deletes a value from set**

Ex:

```
<script>
        var set = new Set();
        set.add( 10 );
        set.add( 20 );
        set.add( 30 );
        set.add( 10 ); //we cant add duplicate values in set
        console.log( set );
        console.log( set.size );
        console.log( set.has( 20 ) );
    </script>
```



➢ for of Control Structure:

A predefined looping control structure through which we could able to iterate over a **SET or MAP** data structure.

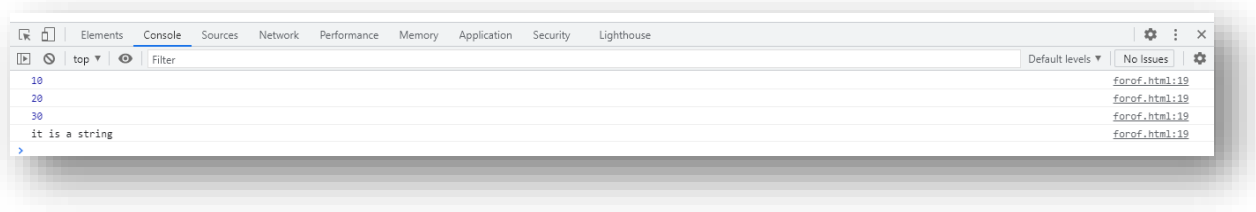
Syntax: for (var tempData of set.values()) {

….

}

Ex:

```
<script>
        var set = new Set();
        set.add( 10 );
        set.add( 20 );
        set.add( 30 );
        set.add( "it is a string" );
        for ( var tempData of set.values() ) {
            console.log( tempData );
        }
    </script>
```



➢ Map Data Structure:

A predefined data structure almost like an JSON object used to hold the data in the form of key value pairs.

Syntax: var demo = new Map( );

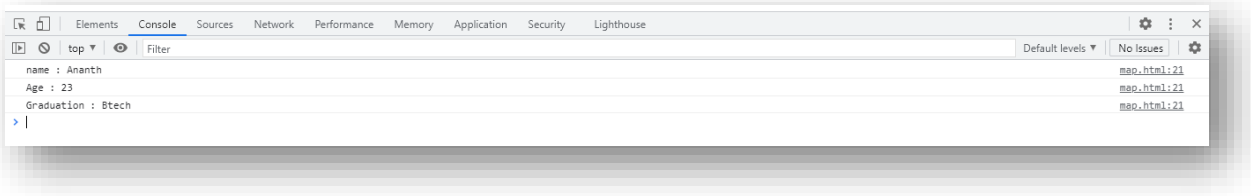
- **Following are predefined methods can be applied on Map,**
  - **set("key","value"); // to add value with corresponding key.**
  - **get("key") //throws corresponding value to a particular key.**
  - **entries ( ) //return values with a map.**
  - **Clear( ) //clears data from MAP**
  - **Delete ("key") //deletes corresponding key and its value from map**

Syntax to iterate Map Data Structure:

```
for (var [key,index] of map.entries()) {
        ……;
};
```

Ex:

```
<script>
        var demo = new Map();
        demo.set( "name", 'Ananth' );
        demo.set( "Age", 23 );
        demo.set( "Graduation", 'Btech' );

        //console.log( demo );

        for ( var [ key, value ] of demo.entries() ) {
            console.log( `${ key } : ${ value }` );
        }
    </script>
```



➢ JavaScript Promises:

A new topic been supported from ECMA-6, promise is an object which represents the eventual, successful compilation or failure of a asynchronous operation and its corresponding value.

- Every promise takes a resolver and rejection call back function which gets invoked based on success or error.
- **"promise"** is a predefined class through which we can create any number of promises with in a single page application (SPA).

  Syntax: var promise =

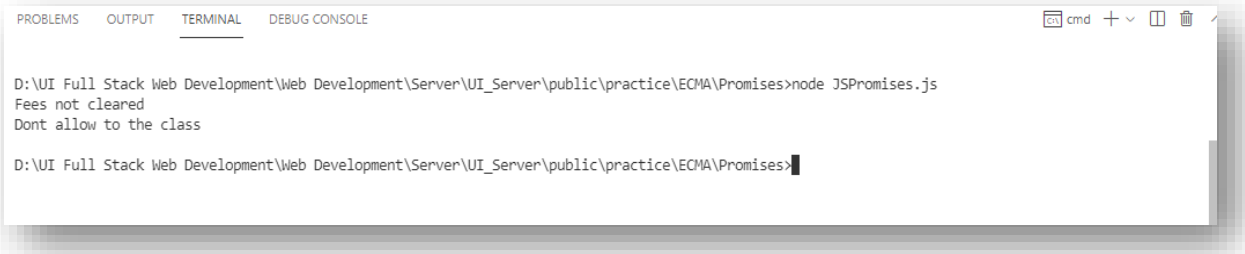
  new Promise((success resolver callback, rejectorCallback) => {

  });

- **Invoking Promise:**
  Method(<optionalparam>).then(successcallback).catch(error callback);

Ex:

```
var isFeePaid = false;
var doJob = () => {
     /*if ( isFeePaid ) {
          console.log( 'allow to access the notes' );
     } else {
          console.log( 'ask to clear the payment' )
     }*/
     var myTask = new Promise( ( resolve, reject ) => {
          if ( isFeePaid ) {
               resolve( 'allow to enter the Class' );
          } else {
               reject( 'Dont allow to the class' );
          }
     } );
     myTask.then( ( value ) => {
          console.log( 'Fees Cleared' );
          console.log( value );
     }, ( error ) => {
          console.log( 'Fees not cleared ' );
          console.log( error );
     } )
}
doJob();
```



➢ Async and Await:

More recent additions to the JavaScript language are async functions and the await keyword, added in ECMAScript 2017. There are two parts to using async/await in your code.

- **The Async keyword:**

  First of all, we have the async keyword, which you put in front of a function declaration to turn it into an async function. An async function is a function that knows how to expect the possibility of the await keyword being used to invoke asynchronous code. The async keyword is added to functions to tell them to return a promise rather than directly returning the value.

- **The await keyword:**

The advantage of an async function only becomes apparent when you combine it with the await keyword. await only works inside async functions within regular JavaScript code, however it can be used on its own with JavaScript modules.

await can be put in front of any async promise-based function to pause your code on that line until the promise fulfils, then return the resulting value.

Ex:

```
<script>
        var showAccountClosingBalance = ( balance ) => {
            alert( 'The account is closed with balance with : ' + balance );
        }
        var getActualBalance = async () => {
            var balance = 0;
            var myPromise = new Promise( ( resolve, reject ) => {
                setTimeout( () => {
                    balance = 1500;
                    resolve( balance );
                }, 5000 );
            } );
            balance = await myPromise;
            showAccountClosingBalance( balance );
        }
        getActualBalance();
    </script>
```

To get More information about web development concepts check MDN

https://developer.mozilla.org/en-US/

--Front-End Completed--