Front End Development Course Training Material

What Does Front Developer Do ?

A front-end developer is a type of software developer who specializes in creating and designing the user interface (UI) and user experience (UX) of websites and web applications. The primary responsibility of a front-end developer is to ensure that the visual and interactive aspects of a website or application are user-friendly, aesthetically pleasing, and functionally efficient.

A front-end developer builds the front-end portion of websites and web applications—the part users see and interact with. A front-end developer creates websites and applications using web languages such as HTML, CSS, and JavaScript that allow users to access and interact with the site or app.

Roles and Responsibilities :-

1. **User Interface Design**
2. **Web Development Languages**
3. **Responsive Design**
4. **Performance Optimization**
5. **Front end frameworks and Libraries**
6. **Version Control**

Introduction Web Programming

Web programming refers to the development of web applications and websites that are accessed over the internet . Basically it involves creating web pages , web applications and other online content that are displayed over web browsers . The programming languages that

are involved in web programming are Html , CSS , Javascript , Perl , Php etc . Each language has its own strength and weakness , the choice of language needed depends on the requirements of the project .

Key components of web programming

1. **Client side Technologies :-**

- **HTML :-** HTML stands for HyperText Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within a tag which defines the structure of web pages.
- **CSS :-** css is a stylesheet language used to design the webpage to make it attractive. The reason for using CSS is to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes up each web page.
- **Javascript :-** is the most powerful and versatile programming language used on the web. It is a lightweight, cross-platform, single-threaded and interpreted programming language. It is a commonly used programming language to create dynamic and interactive elements in web applications, easy to learn, compiled language.

2. **Server Side Technologies :-**

- **Server-Side Languages**:- Languages like PHP, Python, Ruby, Java, and Node js are used to handle server-side logic, process requests, and generate dynamic content.
- **Databases:-** Systems like Mysql, Postgres, Mysql or SQLite are used to store and retrieve data dynamically, enabling web applications to manage and manipulate information.
- **Server Environment:-** Software environments such as Apache, Nginx, or Microsoft IIS provide the infrastructure to host and serve web applications. They handle incoming requests, route them to the appropriate handlers, and send responses back to clients.

3. Frameworks :-

- **What is a Framework ?**

  Framework is a pre-written collection of standardized Html , CSS and Javascript code that developers can use to build web applications more efficiently .

  Basically it provides a set of tools , conventions , libraries for building user interfaces , handling events , managing application states , and interacting with APIs .

  Why do we need a Framework ?

They help developers to organize the code , reduce repetitive work , and achieve consistency across the project .

What is a Library ?

Library is a set of pre - written code used to perform a specific task , it looks after the details , offers functions .

Some popular Frameworks and libraries that a frontend developer must know are as follows

- **React :- React is a Javascript library , owned by facebook released in 2013 .**
- **Tailwind CSS :- Tailwind Css is a Css framework , it is highly customizable and provides a wide range of configurations .**
- **Angular :- Angular is based on Typescript ,it's a Javascript framework and superset of Javascript.**
- **Vue.js :- Vue.js is a Javascript framework , MVVM Model (** Model - View - ViewModel)
- **Bootstrap :- Bootstrap is an open source CSS framework , developed by Twitter .**

- **Material Design Lite :- It's a framework developed by Google , that is based on Material Design**
- **Material UI :- It is a design language developed by Google in 2014 , it uses more grid based layouts , responsive animations . (react library that implements google material design)**

While Material Design Icons offer a comprehensive set of guidelines and components that can be customized to fit specific needs, Material Design Lite provides a more limited range of customization options, focusing on simplicity and ease of use for web developers.

HTML Content :-

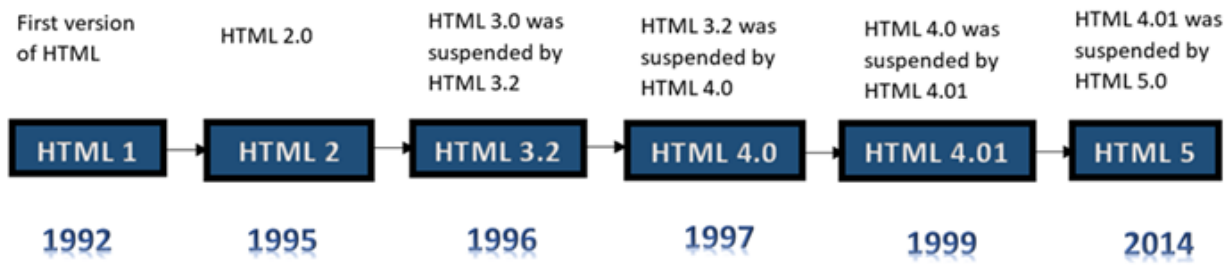
Introduction to web programming and html (history, syntax , concepts involved in html)
Html Basic formatting Tags (head , body , style , paragraph etc) , Atributes
Grouping and using span and div tags
Lists (ordered , unordered list)
Images , Hyperlink (anchor tags)
Tables
iFrames , intro to forms
Forms , Headers
Html Miscellaneous (Meta tags and depreciated tags of HTML along with XHTML and attributes)
Web browser
HTML 5 introduction
Semantics
Audio and video support
Canvas Elements

Geolocation API
Local storge
Responsive Images
Drag and Drop
Web workers , web sockets
Form Enhancements

HTML (Hypertext Markup language)

History

Tim Berners -Lee invented HTML in 1989 and officially introduced it to the world in the early 1990s. In late 1994, he founded the World Wide Web Consortium (W3C), which took over the HTML specifications. After several years of working on the specifications, the W3C switched its focus from HTML to XHTML.



HTML stands for **HyperText Markup Language**, it is a Standard Markup language for web pages. HTML is used to create content and structure of any web page. If you think of the human body as a

web page then HTML is the skeleton of the body. It is the building block of web pages .

HTML Basics: In the basic part we have covered all the fundamentals of HTML like - editors, basic tags, elements, attributes, heading, paragraph, formatting, etc..

HTML Tables: After getting the knowledge of fundamentals of HTML we should learn about tables. The primary tags used to create tables are <table>, <tr>, <td>, and <th>.

HTML Lists: The lists can be ordered or unordered depending on the requirement. In html we can create both ordered and unordered lists by using <ol> and <ul> tags and for the list item we can use <li> tag.

HTML Links: Links allow visitors to navigate between Web sites by clicking on words, phrases, and images. A link is specified using HTML tag <a>.

HTML Backgrounds: Background of a web page is a layer behind its content, which includes text, images, colors and various other elements. HTML allows you to change the background color of any elements within a document using bgcolor attribute.

HTML Colors: Colors are a way of specifying the appearance of web elements. Colors are very important aspects of web

design, as they not only enhance the visual appeal but also influence user behavior. They are also used to evoke emotions and highlight important content.

HTML Form: HTML forms are simple forms that have been used to collect data from the users. HTML form has interactive controls and various input types such as text, numbers, email, password, radio buttons, checkboxes, buttons, etc. There are lots of form elements in HTML, you can learn those from HTML - Forms.

HTML Media: Media is an important element in HTML, sometimes we want to include videos and audios into our websites, we can embed any media into our websites.

HTML Header: Header part of an HTML document is represented by the <head> tag. It serves as a container of various other important tags like <title>, <meta>, <link>, <base>, <style>, <script>, and <noscript> tags.

HTML Layout: Layouts specify the arrangement of components on an HTML web page. A good layout structure of the webpage is important to provide a user-friendly experience on our website. It takes considerable time to design a website's layout with a great look and feel.

HTML Graphics: HTML allows two types of graphics development in the document directly. SVG is an XML-based

markup language used for creating scalable 2D graphics and graphical applications, and Canvas gives you an easy and powerful way to draw graphics using JavaScript.

HTML Applications

As mentioned before, HTML is one of the most widely used languages on the web.

Web Page Development: HTML is used to create pages that are rendered over the web. Almost every page of the web has HTML tags in it to render its details in the browser.

Responsive UI: HTML pages now-a-days works well on all platforms, mobile, tabs, desktop or laptops owing to responsive design strategy.

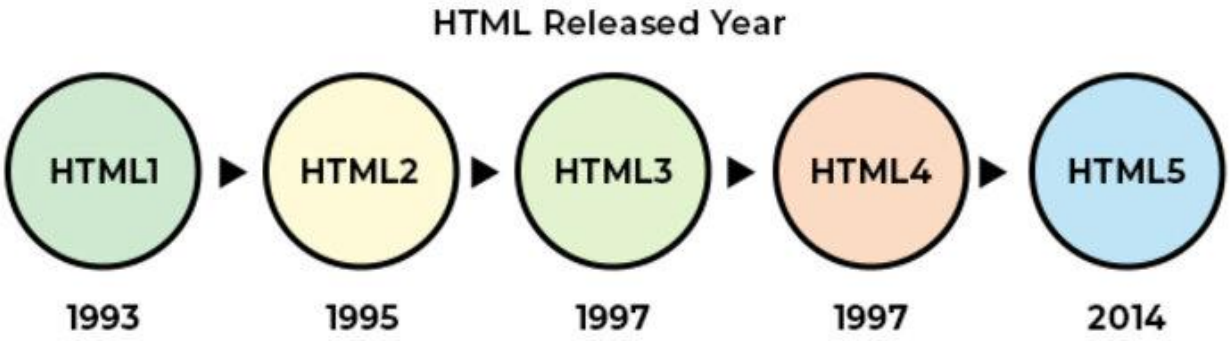
Game Development: HTML5 has native support for rich experience and is now useful in the gaming development area as well.

Mobile application development: HTML with CSS3 and Javascript can be used for developing cross-platform mobile applications.

Multimedia and video streaming: HTML5 offers support for multimedia elements like video and audio, which enables seamless media playback directly in web browsers.

Geolocation: Allows websites to request a user's geographic location. This is helpful for location-based applications and services.

HTML Versions



HTML Document Structure

HTML elements are hierarchical, meaning we can create elements inside of an element. But there are few rules to follow like the head can not be placed inside of a body like that so to know the basic structure of an HTML document please check the below example code.

```
<!-- HTML Version Declaration -->

<!DOCTYPE html>

<!-- HTML Root Element -->

<html>
```

```
<!-- HTML Head Section -->
<head>
    <!-- HTML Document Title -->
    <title>This is Title</title>
    <link>
    <meta>
</head>

<!-- HTML Body Section -->
<body>
    <!-- HTML Header Element -->
    <h1>This is Header</h1>
    <!-- HTML Paragrapgh Element -->
    <p>This is a Paragraph</p>
</body>
</html>
```

Role of Web Browsers in HTML

Web Browsers like Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari, Opera, etc are able to show the output of HTML code, it will define the font size, weight, structure, etc based on tags and attributes.

-
- **HTML Tags , Elements and Attributes :-**
- **HTML Tags:** Tags are similar to keywords, which specify how a web browser will format and display content. A web browser can differentiate between simple content and HTML content with the use of tags.
- All HTML tags must be enclosed within < > these brackets.
- Every tag in HTML performs different tasks.
- If you have used an open tag <tag>, then you must use a close tag (except some tags)

  Syntax:- <tag> content </tag>

  Unclosed HTML Tags

  Some HTML tags are not closed, for example br and hr.

  - **
 Tag**: br stands for break line, it breaks the line of the code.
  - **<hr> Tag**: hr stands for Horizontal Rule. This tag is used to put a line across the webpage.

Tags in HTML:-

- HTML Meta Tags -> (DOCTYPE, title, link, meta and style)

- HTML Text Tags -> <p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, , <pre>
- HTML Link Tags -> <a> , <base>
- HTML List Tags -> , <ul> ,
  HTML Attributes: Attributes are used to customize an element's behavior, special terms called HTML attributes are utilized inside the opening tag. An HTML element type can be modified via HTML attributes.
- **Syntax** :- name=”value”

  Ex:- src = “ “ , alt = “ ” , class = “ “ etc

  HTML Elements: Elements are building blocks of a web page. It consists of a start tag, an end tag, and the content between them.

Grouping Elements :- Arranging elements like (images , text , forms , etc) together and applying styling to it using CSS is specified as Grouping of Elements .

Grouping can be performed with the help of various tags such as , , <fieldset > , <section > , <footer> , <header> etc .

- ** :-** The div tag is known as the **Division tag**. The HTML tag is a block-level element used for grouping and structuring content. It provides a container to organize and style sections of a webpage, facilitating layout design and CSS

styling , this tag has both opening( ) and closing ( ) tags and it is mandatory to close the tag.

Ex :-

```
<!DOCTYPE html>

<html>


<head>
    <title>Div tag</title>


    <style>
        div {
            color: white;
            background-color: #009900;
            margin: 2px;
            font-size: 25px;
        }
    </style>
</head>


<body>
    <div> div tag
        <div>div 2</div>
    </div><br>
```

```
    <div> div tag </div><br>

    <div> div tag </div><br>

    <div> div tag </div><br>
</body>
</html>
```

- ** :-** The HTML span element is a **generic inline container** for inline elements and content. It used to group elements for styling purposes (by using the class or id attributes).

  A better way to use it when no other semantic element is available. The span tag is very similar to the div tag, but div is a block-level tag and span is an inline tag.

  Ex:-

```
<!DOCTYPE html>
<html>
<head>
    <title>span tag</title>
</head>
<body>
    <h2>Welcome To GFG</h2>
```

```
<p>
        <span style="background-color:lightgreen">
            Front End Training
        </span>
        starts from June and will end in september
        <span style="color:blue;">
            Classes will be conducted in
        </span> your own
        <span style="background-color:lightblue">
            shift timings
        </span>
        try to attend classes regularly
    </p>
</body>
</html>
```

Difference between and

1. is block level element , while < span > is inline level element .
2. accepts align attribute while < span > does not accept align attribute.
3. This tag should be used to wrap a section, for highlighting that section , < span > . This tag should be used to wrap any specific word that you want to highlight in your webpage.

Fieldset tag :- It is used to group the related elements in the form and it draws the box around the related elements . It can be used anywhere in html , but except button tag .

The main difference between div and fieldset tag is that div is used for layout structure in webpage , whereas fieldset is used for layout specifically for form fields .

Ex :-

```
<!DOCTYPE html>
<html>
<body>

<h1>The fieldset element</h1>

<form >
 <fieldset>
  <legend>Personal</legend>
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email"><br><br>
  <label for="birthday">Birthday:</label>
  <input type="date" id="birthday" name="birthday"><br><br>
  <input type="submit" value="Submit">
 </fieldset>
</form>
```

```
</body>
</html>
```

Footer tag :- Footer tag is used to define a footer for a document or section . <footer> element contains information such as

- Copyright information
- Authorship information
- Contact information
- Back to top links
- Related documents

You can have several footer elements in one document

Section tag:- It defines a section in a document

Legend tag :- It defines a caption for Fieldset element

Lists :-

List is a record of related information , used to display the data or any information on web pages in **ordered** or **unordered** form .

Lists are of three types

- Ordered List
- Unordered List or Bulleted List
- Description List

Ordered List :- In ordered list all the items are marked by number by default sequentially . It starts <ol> tag and items inside it as written within <li> tags and every tag must be closed .

The elements within the list tag <ol> are referred to as items .

 → ordered list

 → items of list

Ex:-

```
<html>
    <head>
        <Title>Ordered List</Title>
    </head>
    <body>
        <ol>
            <li>Java</li>
            <li>Python</li>
            <li>Ruby</li>
            <li>React</li>
            <li>JavaScript</li>
            <li>Nodejs</li>

        </ol>
    </body>
</html>
```

Unordered List:- The list items here are marked with Bullets . It starts with <ul> tag , each list item starts with <li> tag .

Ex:- Replace in the above code with below part

If you want different shape types then use

Style → `<ul style = "list-style-type:square">`

Type → `<ul type="circle">`

```
<ul>
            <li>Css</li>
            <li>Material UI</li>
            <li>Tailwind CSS</li>
            <li>Bootstrap</li>
            <li>Next.Js </li>
            <li>Nodejs</li>
</ul>
```

Description List :- Description List is a list of terms with a description of each term , <dl> tag defines description list , <dd> tag describes each term .

Ex:-

```
<dl>
            <h1>Description List</h1>
            <dt>Java</dt>
            <dd> - Programming Language </dd>
```

```
        <dt>Python</dt>
        <dd> - Programming Language </dd>
        <dt>JavaScript</dt>
        <dd> - Programming Language </dd>
        <dt>React</dt>
        <dd> - Framework </dd>
    </dl>
```

Replace the codes according to the list types

Task :-

1. **What are iFrames ?**
2. **Purpose of iFrames ?**
3. **In Which case we use iFrames ?**

Image tag :- Image tag is used to display the images in a webpage by linking them . It is defined by attributes like src , width , height , cross-origin alt etc and does not have any closing tag .

There are 2 ways of inserting an image into webpage

- By providing a full path or address (URL) to access an internet file.
- By providing the file path relative to the location of the current web page file.

  Syntax :-

```
<img src="url" alt="some_text" width="" height="">
```

Ex:-

```
<html>



<head>

    <title>Welcome To FrontEnd Development</title>

</head>

<body>

    <h2>NextJs Developer</h2>

<img src=”use your own images“  alt="Next js img" />

</body>

</html>
```

Adding image in animated format in HTML :-

To add an animated image in HTML, use the <img> tag with the src attribute pointing to a GIF file, providing engaging motion to enhance webpage content.

Ex:-

```
<html>
```

```
<body>

    <h3>Adding a gif file on a webpage</h3>

    <img
src="https://media.geeksforgeeks.org/wp-content/uploads/20210203164653/ezgifcomgifmaker2.gif"

         alt="smiley"

         style="width: 200px; height: 200px" />

</body>

</html>
```

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. This attribute determines where the user is directed upon clicking the link.

Syntax:- `<a href = "link"> Link Name </a>`

Ex:-

```
<html>
```

```
<body>

    <h2>

        Welcome to GeeksforGeeks

        HTML Tutorial

    </h2>

    <!-- Opening link in same tab  -->

    <a href="https://www.geeksforgeeks.org/html-tutorials/">

        <!-- Opening Link in new tab  -->

    <a href="https://www.geeksforgeeks.org" target="_blank">Visit
GeeksforGeeks</a>

        GeeksforGeeks HTML Tutorial

    </a>

</body>

</html>
```

Linking Image with anchor tag :-

When you click on the image you will be redirected to new page .

Ex:-

```
<html>
```

```
<body>

    <p>Click on the image to open web page.</p>



    <!-- anchor tag starts here -->

    <a href="https://www.geeksforgeeks.org/">

        <img src=”use img from google”  width="300" height="250" />

    </a>

    <!-- anchor tag ends here -->

</body>

</html>
```

Task - 2 ➖

1. **Create an unordered list inside ordered list , in unordered list the items must be marked by using square points .**
2. **Take 4 sections in a webpage and display images inside it .**

Tables :-

Tables are a way to represent the data in form of rows and columns in tabular format. We can store text and numerical information data in table. A table is a useful tool for quickly and easily finding

connections between different types of data. Tables are also used to create databases.

Tags used in HTML tables are as follows :-

1. **<table> :-** Defines structure for organizing data rows and columns within a webpage .
2. **<tr> :-** Represents a row within an HTML table, containing individual cells.
3. **<th> :-** Shows a table header cell that typically holds titles or headings.
4. **<td> :-** Represents a standard data cell, holding content or data.
5. **<captain> :-** Provides a title or description for the entire table.
6. **<thead> :-** Defines the header section of a table, often containing column labels.
7. **<tbody> :-** Represents the main content area of a table, separating it from the header or footer.
8. **<tfoot> :-** Specifies the footer section of a table, typically holding summaries or totals.
9. **<col> :-** Defines attributes for table columns that can be applied to multiple columns at once.
10. **<colgroup> :-** Groups together a set of columns in a table to which you can apply formatting or properties collectively.

Defining Tables in HTML:-

An HTML table is defined with the “table” tag. Each table row is defined with the “tr” tag. A table header is defined with the “th” tag. By default, table headings are bold and centered. A table data/cell is defined with the “td” tag.

Table Cells:- Table Cell are the building blocks for defining the Table. It is denoted with <td> as a start tag & </td> as a end tag.

Syntax:- <td> content </td>

Table Rows:- The rows can be formed with the help of combination of Table Cells. It is denoted by <tr> and </tr> tag as a start & end tags.

Syntax:-

<tr> Content...</tr>

Table Headers:- The Headers are generally use to provide the Heading. The Table Headers can also be used to add the heading to the Table. This contains the <th> & </th> tags.

Syntax:-

<th> Content...</th>

Ex:-

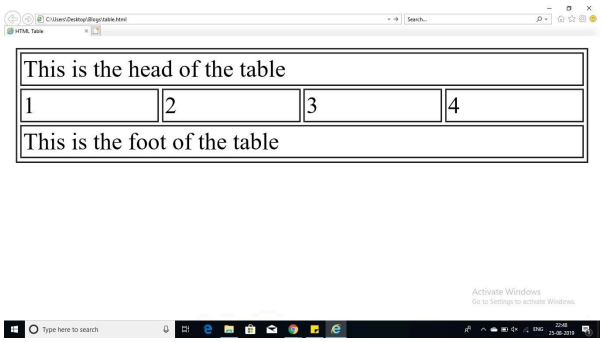
```
<!-- index.html -->

<!DOCTYPE html>

<html>



<head>

    <style>

        table,

        th,

        td {

            border: 1px solid black;

        }

    </style>

</head>



<body>
```

```
<table style="width:100%">
  <tr>
    <th>First Name</th>
    <th>Last Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Priya</td>
    <td>Sharma</td>
    <td>24</td>
  </tr>
  <tr>
    <td>Arun</td>
    <td>Singh</td>
    <td>32</td>
```

```
        </tr>
        <tr>
            <td>Sam</td>
            <td>Watson</td>
            <td>41</td>
        </tr>
  </table>
</body>
</html>
```

Task :-

Write the code for the following format.



Forms :-

Form is used to collect the inputs from the user via variety of interactive controls. These controls range from text fields, numeric inputs, email fields, passwords fields, check boxes, radio buttons, submit buttons . Html forms serves as versatile container to collect input from users and there by enhancing user interaction.

Syntax:-

<form>

// form elements

</form>

Form Elements :-

Form comprises several elements each justifies unique purpose. Among all the elements <input> element is versatile as it accepts input data from users of various types such as text, password, numbers, email etc.

Let us see the elements of forms

1. Label :- it defines the label for form elements
2. Input :- takes input from the user of various types such as text, numerical, password, email etc.
3. Button :- It defines a clickable button to control other elements or execute a functionality.
4. Select :- It is used to create a drop-down list.
5. Textarea:- It is used to input long text content.
6. Fieldset:- It is used to draw a box around other form elements and group the related data.
7. Legend :- It defines a caption for fieldset elements
8. Option :- It is used to define options in a drop-down list.
9. Optgroup :- It is used to define group-related options in a drop-down list.
10. Datalist:-It is used to specify predefined list options for input control.

Ex :-

```
<!DOCTYPE html>
```

```
<html lang="en">

<head>

<title>Html Forms</title>

</head>

<body>

  <h2>HTML Forms</h2>

  <form>

    <label for="username">Username:</label><br>

    <input type="text" id="username" name="username"><br><br>



    <label for="password">Password:</label><br>

    <input type="password" id="password"
name="password"><br><br>



    <input type="submit" value="Submit">

  </form>
```

```
</body>

</html>
```

Ex :-

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Form</title>
    <style>
      body {
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh;
        margin: 0;
        background-color: #f0f0f0;
      }
```

```
form {
    width: 400px;
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px
        rgba(0, 0, 0, 0.1);
}

fieldset {
    border: 1px solid black;
    padding: 10px;
    margin: 0;
}

legend {
    font-weight: bold;
    margin-bottom: 10px;
```

```
}

label {
    display: block;
    margin-bottom: 5px;
}

input[type="text"],
input[type="email"],
input[type="password"],
textarea,
input[type="date"] {
    width: calc(100% - 20px);
    padding: 8px;
    margin-bottom: 10px;
    box-sizing: border-box;
    border: 1px solid #ccc;
    border-radius: 4px;
```

```
        }

        input[type="radio"] {
            margin-left: 20px;
        }

        input[type="submit"] {
            padding: 10px 20px;

            border-radius: 5px;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <form>
        <fieldset>
            <legend>
```

```
    User personal information
</legend>
<label
    >Enter your full name</label
>
<input type="text" name="name" />
<label>Enter your email</label>
<input
    type="email"
    name="email"
/>
<label>Enter your password</label>
<input
    type="password"
    name="pass"
/>
<label
    >Confirm your password</label
```

```
>
<input
    type="password"
    name="confirmPass"
/>
<label>Enter your gender</label>
<input
    type="radio"
    name="gender"
    value="male"
/>Male
<input
    type="radio"
    name="gender"
    value="female"
/>Female
<input
    type="radio"
```

```
        name="gender"
        value="others"
      />Others
      <label
        >Enter your Date of
        Birth</label
      >
      <input type="date" name="dob" />
      <label>Enter your Address:</label>
      <textarea
        name="address"
      ></textarea>
      <input
        type="submit"
        value="submit"
      />
    </fieldset>
  </form>
```

```
  </body>
</html>
```

Task :-

Design a form to place an order for a product, specify all the details of the product, and take a reference of any Ecommerce platform.

Nested table (sample code)

```
<body>
  <h4>How to create nested tables within tables in HTML?</h4>
  <table class="main-table">
    <tr>
      <td>
        Main table cell 1
        <table class="nested-table">
          <tr>
            <td>Nested table cell 1</td>
            <td>Nested table cell 2</td>
          </tr>
        </table>
```

```
        </td>
        <td>Main table cell 2</td>
      </tr>
    </table>
  </body>
```

Iframes :-

Iframes are used to display a webpage within a webpage .

It specifies inline frame

Syntax:-

```
<iframe src="url" title="description"></iframe>
```

- The **src** attribute specifies the **URL** of the document you want to embed.
- Iframes can include **videos**, **maps**, or **entire web pages** from other sources.

Ex:-

```
<html>
```

```
<body>



<h2>HTML Iframes</h2>

<p>You can use the height and width attributes to specify the size of the iframe:</p>



<iframe src="demo_iframe.htm" height="200" width="300" title="Iframe Example"></iframe>

</body>

</html>
```

Ex :- Embed a map

```
<html>

    <body>

        <h1>Map</h1>

        <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3807.01379096796!2d78.54706697414181!3d17.411125802122857!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3
```

```
m3!1m2!1s0x3bcb9963b2a5d0a7%3A0x1f10a3b1a2eef974!2sMantha%20Tech%20Solutions%20
Private%20Limited!5e0!3m2!1sen!2sin!4v1718270997552!5m2!1sen!2sin" width="600"
height="450" style="border:0;"

         allowfullscreen="" loading="lazy"
referrerpolicy="no-referrer-when-downgrade"></iframe>

    </body>

</html>
```

Ex:- Embed a video

```
<html>

    <body>

        <h1>Youtube Video </h1>

        <iframe width="560" height="315"
src="https://www.youtube.com/embed/SAqi7zmW1fY?si=hqPpezPKezbfKnac"

         title="YouTube video player" frameborder="0" allow="accelerometer;
autoplay; clipboard-write;

         encrypted-media; gyroscope; picture-in-picture; web-share"

         referrerpolicy="strict-origin-when-cross-origin"
allowfullscreen></iframe>
```

```
    </body>

</html>
```

Quotation Tag :-

Html quotation elements are used to insert quoted text in a webpage . It means some part of text that is different from the rest of the portion .

Some quotation elements elements are as follows

- <abbr> :- defines abbreviation or acronym .
- <address> :- defines contact info for the owner of the document.
- <bdo> :- defines text direction (right to left or left to right) .Bi-directional override
- <q> :- defines shortline quotation , enclosed with quotation marks .

Ex:-

```
<html>

    <body>

        <h1>Quotation tag </h1>

        <p>React is Javascript library </p> <br>
```

```
        <p>

            <bdo dir="rtl">

                Angular is Javascript Framework

            </bdo>

        </p>

        <br>

        <abbr title="Spring tool suite">STS</abbr>

        <blockquote>

            <p>

                Learning Management system app includes Java , Aws ,
Devops course

            </p>

        </blockquote>

       <p>The best platform to learn react is <cite>Geeks for
Geeks</cite> start your course today and excel in Front

    end development</p>
```

```
        <address>
            <p>
                Address:<br />
                H.No.- 3-8-876, Uppal
                Park,<br />
                Sector-142, Noida Uttar Pradesh -
                201305
            </p>
            <q>Advanced version of react is NextJs</q>
        </address>
    </body>
</html>
```

Marquee Tag :- Marquee tag creates scrolling text or image effect in a webpage . It allows content to move horizontally or vertically across the screen, providing a simple way to add dynamic

movement to elements. It includes attributes like direction to specify whether the content moves left, right, up, or down.

Marquee tag is deprecated in HTML 5

Syntax:-

```
<marquee>

 Content

</marquee>
```

Ex:-

```
<html>

    <body>

        <body>

            <div class="main">

                <marquee

                        direction="left" loop="">
```

```
                <div >

                    GeeksforGeeks

                </div>

                <div >

                    A computer science portal for geeks

                </div>

            </marquee>

        </div>

    </body>

</html>
```

Semantic Elements :- A semantic element clearly describes its meaning to both the browser and the developer.

Examples of non-semantic elements: and - Tells nothing about its content.

Examples of semantic elements: <form>, <table>, and <article> - Clearly defines its content.

Let us see some semantic elements used in a webpage

- <article>
- <aside>
- <details>
- <figcaption>
- <figure>
- <footer>
- <header>
- <main>
- <mark>
- <nav>
- <section>
- <summary>
- <time>

Ex :-

```
<html>

    <head>

        <style>

            aside {

                width: 50%;

                padding: 15px;

                margin: 15px;

                color: black;

                background-color: blanchedalmond;
```

```
                float: right;



            }
        </style>
    </head>
    <body>
        <article>
        <p>Used by some of the world's largest companies,
             Next.js enables you to create high-quality web
applications with the power of
              React components. Used by some of the world's largest
companies,
              Next.js enables you to create high-quality web
applications with the power of
               React components</p>
            </article>
        <aside>
            Used by some of the world's largest companies,
             Next.js enables you to create high-quality web
applications with the power of React components.
        </aside>
        <abbr title="World Wide Web consortium">W3C</abbr>
        <address>
            <p>Address <br>
            H.No. 5-8-887, VNC Colony <br>
```

```
        Uppal , Hyderabad</p>
        </address>

        <details>
            <summary>Components</summary>
            <p>
                Breaking down in to small modules are referred as
components
                Breaking down in to small modules are referred as
components
                Breaking down in to small modules are referred as
components

            </p>
        </details>
    </body>
</html>
```

Aside :- aside tag is used to display a content that is different or is separated from the main content .

Article :- article tag is used to define an article in a web page . In real time we use article tags to define blog posts , news articles etc .

Details :- It specifies additional details that the user can open and close on demand , basically details and summary tag are used together .

Summary :- Summary tag specifies heading for the details tag , it hides or displays text based on user demand .

As mentioned above we use summary tag for heading , when user clicks on that it hides and shows the details of that , for details we use details tag . This is mentioned as an example in the above code .

Address :- address tag is used to display the address when we use this tag , it shows the address in italic font by default .

Footer :- footer tag is used to define footer .

Footer tag is explained in the Grouping concept ,check the reference form there .

Task :-

Design a web page that embeds a map , video in four different sections

(take four sections , and embed a map in first , video in second follow the same pattern for next sections)

Design a webpage with following requirements

- **Navbar in header section**
- **Home , Contact , About options (pages)**
- **Design each page separately .**
- **In Home page display content along with images**
- **In Contact page display contact details along with location**
- **In About page display an article about the web page**
- **Also include footer section**

- **Display the list of items in the table or list according to the topics you choose in about page after article .**
- **Topics :- Training Institute for online courses ,**

Or

Groceries Delivery site

Choose any one topic according to your convenience

- **Also mention the price of course and grocery items if you are using table**

Note :- Remember to use all the concepts and tags of what we discussed so far . All the basics of HTML

Do the above task on Tuesday , and submit it on Wednesday

HTML 5

Overview :-

HTML5 provides details of all 40+ HTML tags including audio, video, header, footer, data, datalist, article etc. This HTML tutorial is designed for beginners and professionals.

HTML5 is the next version of HTML. Here, you will get some brand new features which will make HTML much easier. These new introducing

features make your website layout clearer to both website designers and users. There are some elements like <header>, <footer>, <nav> and <article> that define the layout of a website.

Why HTML5 ?

It is enriched with advanced features which makes it easy and interactive for designer/developer and users.It allows you to play a video and audio file.

It allows you to draw on a canvas.

It facilitates you to design better forms and build web applications that work offline.

It provides you advanced features for that you would normally have to write JavaScript to do.

The most important reason to use HTML 5 is, we believe it is not going anywhere. It will be here to serve for a long time according to W3C recommendation.

HTML5 Attributes :-

Some attributes are defined globally and can be used on any element, while others are defined for specific elements only. All attributes have a name and a value and look like as shown below in the example.

Ex :-

```
<div class = "example">...</div>
```

Attributes may only be specified within **start tags** and must never be used in **end tags**.

HTML5 attributes are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase.

Standard Attributes :-

- **align :-** horizontally aligned tags
- **background :-** places a background image behind an element
- **class :-** Classifies an element for use with cascading style sheets
- **Contenteditable :-** Specifies if the user can edit the content or not
- **draggable :-** Specifies whether or not a user is allowed to drag an element .
- **height :-** Specifies the height of a table , image or table cells .
- **hidden :-** Specifies whether the element should be visible or not
- **id :-** Names an element for use with CSS
- **Item :-** used to group elements
- **Itemprop :-** used to group items
- **spellcheck :-** Specifies if the element must have its spelling or grammar checked.
- **style :-** specifies an inline style for an element
- **valign :-** aligns elements vertically
- **width :-** specifies width of tables , table cells , images etc

Events :-

When users visit your website, they perform various activities such as clicking on text and images and links, hover over defined elements, etc. These are examples of what JavaScript calls **events**.

We can write our event handlers in Javascript or VBscript and you can specify these event handlers as a value of event tag attribute. The HTML5 specification defines various event attributes as listed below −

We can use the following set of attributes to trigger any **javascript** or **vbscript** code given as value, when there is any event that takes place for any HTML5 element.

Let us see some event attributes that are mostly used with HTML5

- **onoffline :-** triggers when document goes offline
- **onabort :-** triggers on an abort event
- **onchange :-** triggers when an element changes
- **onclick :-** triggers on a mouse click
- **ondblclick :-** triggers on mouse double
- **ondrag :-** triggers when an element is dragged
- **ondragend :-** triggers at the end of drag operation
- **ondragstart :-** triggers at the start of drag operation
- **ondragenter :-** Triggers when an element has been dragged to a valid drop target.
- **ondragleave :-** Triggers when an element leaves a valid drop target .
- **ondragover :-** Triggers when an element is being dragged over a valid drop target .
- **ondrop :-** Triggers when dragged element is being dropped .

- **ondurationchange :-** triggers when the length of the media is changed .
- **onemptied :-** Triggers when a media resource element suddenly becomes empty.
- **onended :-** triggers when media has reach end
- **onformchanges :-** triggers when a form changes
- **onforminput :-** triggers when a form gets user input
- **onmousedown :-** triggers when a mouse button is pressed
- **onmousemove :-** triggers when mouse pointer moves
- **onmouseout :-** triggers when the mouse pointer moves out of an element.
- **onmouseover :-** triggers when the mouse pointer moves over an element .

Ex :-

```
<html>

    <body ononline="offline()">

        <h1>Offline example</h1>

        <input type="text" value="hello" onchange="function1()">

        <button onclick="function2()">Submit</button>

         <p ondblclick="function3()">double click this paragraph to call a function</p>



         <p  draggable="true" ondrag="function4()" >drag this paragraph to call a function</p>
```

```
        <script>
            function function1() {
                alert("The input value has changed ");


            }
            function function2() {
                  alert("when you click on button function2 is called and
displayes alert message ");
            }
            function function3() {
                    alert("when you  double click on paragraph element
function3 is called and displayes alert message ");


            }
            function function4() {
                  alert("when you  drag  this  paragraph element function4
is called and displayes alert message ");


            }


          function offline(){
                alert("Browser works in offline mode ");
                console.log("online mode ");
```

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

CSS (Cascading Style Sheets)

Introduction to css , Css Syntax
CSS fonts
CSS colors , css properties
Combinators
Comments , align
Borders , index , lists
Padding , margin
Animation
Media
CSS selectors
CSS text , Display
width , contents , overflow
CSS Rule Set
CSS transforms
Margin

Introduction :-

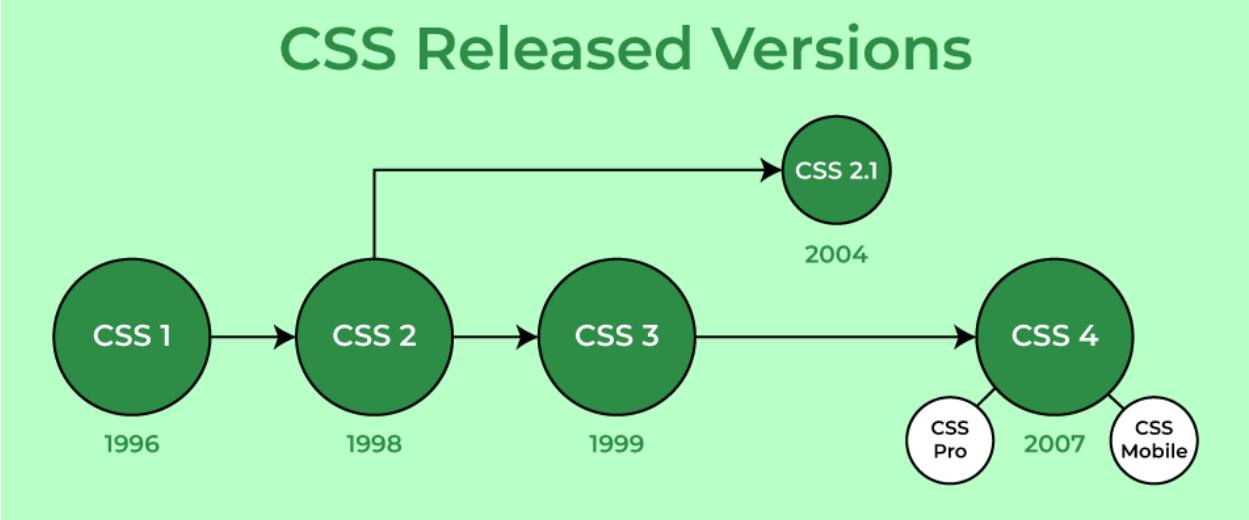
CSS (Cascading Style Sheets) is a language designed to simplify the process of making web pages presentable. It allows you to apply styles to HTML documents, describing how a web page should look by prescribing colors, fonts, spacing, and positioning. CSS provides developers and designers with powerful control over the presentation of HTML elements.

HTML uses tags and CSS uses rulesets. CSS styles are applied to the HTML element using selectors. CSS is easy to learn and understand, but it provides powerful control over the presentation of an HTML document.

Why CSS ?

- **Saves Time:** Write CSS once and reuse it across multiple HTML pages.
- **Easy Maintenance:** Change the style globally with a single modification.
- **Search Engine Friendly:** Clean coding technique that improves readability for search engines.
- **Superior Styles:** Offers a wider array of attributes compared to HTML.

Css Versions :-



Css Syntax :-

Css consists of style rules that are interpreted by the browser and applied to the corresponding elements .

What is a Style Rule Set ?

A Style Rule Set includes a selector and a declaration block

Selector :- targets the specific html element to apply the styles

Declaration :- Combination of a property and its corresponding value

Ex:-

<h1>Hello World</h1>

//Style

h1{ color: “white”;font-size:12px ; } -> style rule set

In above example

h1-> selector (that specifies element to which the style is to be applied)

{ color: “white”;font-size:12px ; } -> this is called as declaration block

It contains one or more declarations each separated by semicolons and it includes css properties in name value format , always css declarations are separated by semicolons and enclosed in curly braces .

Ex :- with css

```
<html>
    <head>
        <style>
            h1{
                text-align: center;
                border: 1px solid black;
                background-color:blue;
                color: aliceblue;
            }
        </style>
```

```
    </head>
    <body>
        <h1>Mantha Tech Solutions</h1>


    </body>
</html>
```

Types of CSS

1. **Inline CSS :-** Inline CSS involves applying styles directly to individual HTML elements using the style attribute. This method allows for specific styling of elements within the HTML document, overriding any external or internal styles.

Ex:-

```
<html>
    <head>
        <style>
            h1{
                text-align: center;
                border: 1px solid black;
```

```
                background-color:blue;
                color: aliceblue;
            }
        </style>
    </head>
    <body>
        <h1>Mantha Tech Solutions</h1>
        <p style="color:black;
                font-style: italic;
                text-align: center;
                font-size: 30px;
        ">We Provide Development and Services.</p>

    </body>
</html>
```

2. **Internal or Embedded CSS :- It** is defined within the HTML document's <style> element. It applies styles to specific HTML elements, The CSS rule set should be within the HTML file in the head section i.e. the CSS is embedded within the <style> tag inside the head section of the HTML file.

Ex:-

Same as previous one

3. **External CSS :-** It contains separate CSS files that contain only style properties with the help of tag attributes (For example class, id, heading, … etc). CSS property is written in a separate file with a .css extension and should be linked to the HTML document using a link tag. It means that, for each element, style can be set only once and will be applied across web pages.

Ex:-

```
<!DOCTYPE html>

<html>



<head>

    <title>External CSS</title>

    <link rel="stylesheet" href="style.css">

</head>



<body>

    <div class="main">

        <div class="sub">We Provide services such as web and mobile app Development</div>
```

```
        <div id="para">
            we follow all the standards of Agile
        </div>
    </div>
</body>


</html>
```

Style.css

```
body {
    background-color:antiquewhite
}

.main {
    text-align: center;
}

.sub {
```

```
    color:aqua;
    font-size: 50px;
    font-weight: bold;
}

#para {
    font-style: bold;
    font-size: 20px;
    color: aqua;
}
```

Important Points :-

- Inline CSS has the highest priority, then comes Internal/Embedded followed by External CSS which has the least priority. Multiple style sheets can be defined on one page. For an HTML tag, styles can be defined in multiple style types and follow the below order.
- As Inline has the highest priority, any styles that are defined in the internal and external style sheets are overridden by Inline styles.

- Internal or Embedded stands second in the priority list and overrides the styles in the external style sheet.
- External style sheets have the least priority. If there are no styles defined either in inline or internal style sheets then external style sheet rules are applied for the HTML tags.

Disadvantages of CSS :-

- **Lack of security :-** Css does not provide security , it is vulnerable to Cross - SIte Scripting attacks (XSS attacks) , be cautious while using css in real time
- **Performance :-** CSS impacts performance of web page badly. It takes more time to load the application or web page because we need to write long lines of codes if we onlyB css for styling .
- **Browser Compatibility :-** CSS renders differently on various browsers it leads to inconsistency in results and outputs , in such case developers need to write browser specific code .

Css Comments :-

Comments are essential for documenting code, providing clarity during development and maintenance. They begin with `/*` and end with `*/`, allowing for multiline or inline annotations. Browsers ignore comments, ensuring they don’t affect the rendering of web pages.

Syntax :-

```
/* content */
```

Comments can be **single-line** or **multi-line**. The **/* */** comment syntax can be used for both single and multiline comments. We may use **<!– –>** syntax for hiding in CSS for older browsers, but this is no longer recommended for use.

Measurement Units in CSS :-

CSS has several units for expressing a length. Many CSS properties take “length” values , such as width , margin, padding, font-size etc

Length is a number followed by a length unit such as 10px, 2em etc

A whitespace cannot appear in between number and unit , if the value is 0 unit can be omitted . For some css properties negative lengths are allowed . There are two types of length ,

1. **Absolute Length :-** The Absolute length units are fixed and a length expressed in any of these will appear as exactly that size .

   But these are not recommended to use on screen , bucz screen sizes vary so much these can be used if the output medium is known before such as print layout .

   Units are cm, mm , in , px* , pt , pc

   1pt = 1/72 inch

2. **Relative Length :-** Relative length units specify a length relative to another length property . These units scale better between different mediums .

   Units are

   em - > 2em means 2 times of the size of current font

   rem - > relative to the font size of the root element .

   % - > relative to the parent element

Difference :-

- Unlike absolute units , relative units are not fixed , their values are relative to another value .
- It means when that other value changes , the relative unit value also changes

CSS Colors :-

CSS Colors are an essential part of web design, providing the ability to bring your HTML elements to life. This feature allows developers to set the color of various HTML elements, including font color, background color, and more.

Color Format :-

Colors in css can be specified by following methods

- Hexadecimal colors

- Hexadecimal colors with transparency
- RGB colors
- RGBA colors
- HSL colors
- HSLA colors

Hexadecimal colors :- A hexadecimal color is specified with: #RRGGBB, where the RR (red), GG (green) and BB (blue) hexadecimal integers specify the components of the color. All values must be between 00 and FF.

For example, the #0000ff value is rendered as blue, because the blue component is set to its highest value (ff) and the others are set to 00.

Hexadecimal color with transparency :-

A hexadecimal color is specified with: #RRGGBB. To add transparency, add two additional digits between 00 and FF.

Ex:- #00ff0080

RGB Colors :-

An RGB color value is specified with the rgb() function, which has the following syntax:

rgb(red, green, blue)

Each parameter (red, green, and blue) defines the intensity of the color and can be an integer between 0 and 255 or a percentage value (from 0% to 100%).

For example, the rgb(0,0,255) value is rendered as blue, because the blue parameter is set to its highest value (255) and the others are set to 0.

Also, the following values define equal color: rgb(0,0,255) and rgb(0%,0%,100%).

Ex :- rgb(0,0,255)

RGBA Colors :-

RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity of the object.

An RGBA color is specified with the rgba() function, which has the following syntax:

rgba(red, green, blue, alpha)

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

Ex :- rgba(0,0,255,0.5)

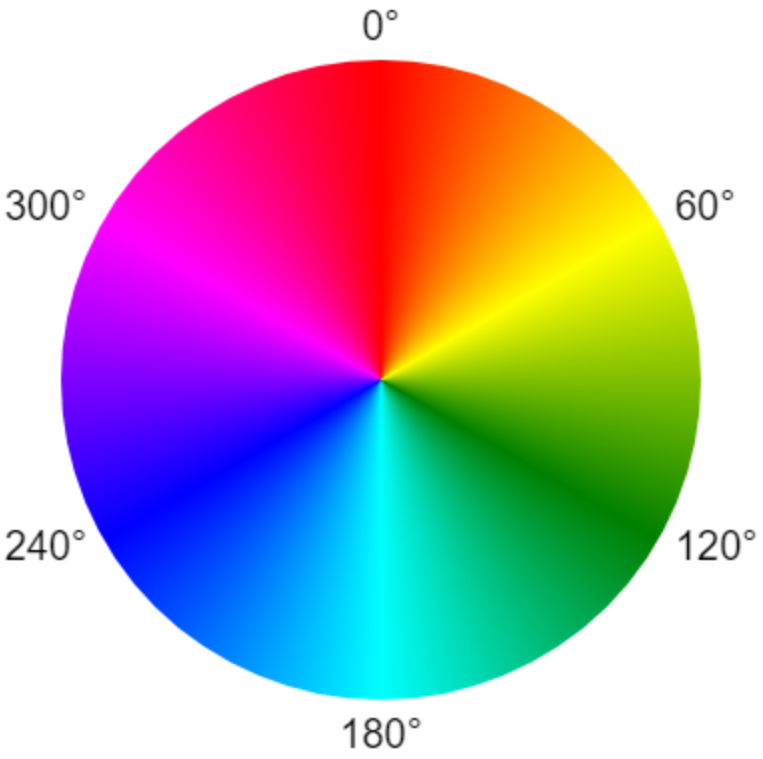
HSL colors :-

HSL stands for hue, saturation, and lightness - and represents a cylindrical-coordinate representation of colors.

An HSL color value is specified with the hsl() function, which has the following syntax:

hsl(hue, saturation, lightness)

Hue is a degree on the color wheel (from 0 to 360) - 0 (or 360) is red, 120 is green, 240 is blue. Saturation is a percentage value; 0% means a shade of gray and 100% is the full color. Lightness is also a percentage; 0% is black, 100% is white.



Ex :-

```
background-color: hsl(0, 100%, 50%);
```

HSLA colors :-

HSLA is an extension of HSL color , and a means alpha parameter is added in this function , its specified as hsla() function .

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

Ex:-

```
background-color: hsla(0, 100%, 50%,0.3);
```

CSS Fonts :-

CSS fonts are used to style the text within HTML elements. The font-family property specifies the typeface, while font-size, font-weight, and font-style control the size, thickness, and style of the text. Combining these properties enhances the typography and readability of web content.

What is typography ?

Typography is the art of and technique of arranging type (text) in a way that is visually appealing and effective for communication .

It involves selecting and using typefaces (fonts) , font size , line spacing , and other text elements to create a clear and readable text .

The following are the types of CSS font properties :-

- CSS font-family property
- CSS font-style property
- CSS font-weight property
- CSS font-variant property
- CSS font-size property
- CSS font-stretch property
- CSS font-kerning property

CSS Margin property :-

The margin property sets the margins for an element, and is a shorthand property for the following properties:

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

If the margin property has four values:

- margin: 10px 5px 15px 20px;
    - top margin is 10px
    - right margin is 5px

    - bottom margin is 15px
    - left margin is 20px

If the margin property has three values:

- margin: 10px 5px 15px;
    - top margin is 10px
    - right and left margins are 5px
    - bottom margin is 15px

If the margin property has two values:

- margin: 10px 5px;
    - top and bottom margins are 10px
    - right and left margins are 5px

If the margin property has one value:

- margin: 10px;
    - all four margins are 10px

Note: Negative values are allowed.

Negative values in margin property :-

It is possible to give margins a negative value. This allows you to draw the element closer to its top or left neighbor , or draw its right and bottom neighbor closer to it .

margin - inline property :-

Css margin property specifies the margin at the start and end in inline direction , shorthand properties for this as as follows

- margin-inline-start
- margin-inline-end

Values for the margin-inline property can be set in different ways:

If the margin-inline property has two values:

- margin-inline: 10px 50px;
    - margin at start is 10px
    - margin at end is 50px

If the margin-inline property has one value:

- margin-inline: 10px;
    - margin at start and end is 10px

Ex:-

```
<html>
    <head>
        <style>
            #p1{
                font-size: 2em;
                color:aliceblue;
                background-color: hsla(0, 89%, 0%,1.0);
                margin:10px;
```

```
}
#p2{
    font-family:'Times New Roman', Times, serif;
    font-size: 70px;
    font-weight: bold;
    font-style: italic;
    font-variant: small-caps;
    font-stretch:condensed;
     margin:-15px;
     margin-left: 200px;

}
div{
    background-color: rgb(255,0,0);
    width:25%;
    height: 200px;
    float: left;

}
#div2{
    background-color: aqua;
    border: solid black 1px;
     margin-inline: 35px;
```

```
        }
    </style>
</head>
<body>
    <p id="p1">HSL Method of applying colors</p>

    <p id="p2">Font properties </p>

    <div >First section</div>

    <div id="div2">Second section</div>

    <div >Third section</div>

</body>
</html>
```

margin-block property :-

The `margin-block` property specifies the margin at the start and end in the block direction, and is a shorthand property for the following properties:

- `margin-block-start`

- `margin-block-end`

Values for the `margin-block` property can be set in different ways:

If the margin-block property has two values:

- margin-block: 10px 50px;
    - margin at start is 10px
    - margin at end is 50px

If the margin-block property has one value:

- margin-block: 10px;
    - margin at start and end is 10px

The CSS `margin-block` and `margin-inline` properties are very similar to CSS properties `margin-top`, `margin-bottom`, `margin-left` and `margin-right`, but the margin-block and margin-inline properties are dependent on block and inline directions.

Note: The related CSS property writing-mode defines block direction. This affects where the start and end of a block is and the result of the margin-block property. For pages in English, block direction is downward and inline direction is left to right.

Padding property :-

An element's padding is the space between its content and its border.

The padding property is a shorthand property for:

- padding-top
- padding-right

- padding-bottom
- padding-left

Note: Padding creates extra space within an element, while margin creates extra space around an element.

This property can have from one to four values.

If the padding property has four values:

- padding:10px 5px 15px 20px;
    - top padding is 10px
    - right padding is 5px
    - bottom padding is 15px
    - left padding is 20px

If the padding property has three values:

- padding:10px 5px 15px;
    - top padding is 10px
    - right and left padding are 5px
    - bottom padding is 15px

If the padding property has two values:

- padding:10px 5px;
    - top and bottom padding are 10px
    - right and left padding are 5px

If the padding property has one value:

- padding:10px;
    - all four paddings are 10px

Note: Negative values are not allowed.

padding-block property :-

An element's padding-block is the space from its border to its content in the block direction, and it is a shorthand property for the following properties:

- padding-block-start
- padding-block-end

Values for the padding-block property can be set in different ways:

If the padding-block property has two values:

- padding-block: 10px 50px;
    - padding at start is 10px
    - padding at end is 50px

If the padding-block property has one value:

- padding-block: 10px;
    - padding at start and end is 10px

padding-inline property :-

An element's padding-inline is the space from its border to its content in the inline direction, and it is a shorthand property for the following properties:

- padding-inline-start

- padding-inline-end

Values for the padding-inline property can be set in different ways:

If the padding-inline property has two values:

- padding-inline: 10px 50px;
    - padding at start is 10px
    - padding at end is 50px

If the padding-inline property has one value:

- padding-inline: 10px;
    - padding at start and end is 10px

Ex :-

```
<html>

    <head>

        <style>

            div{

                background-color: aqua;

                font-size: 2rem;

            }

            .div2{

                border: solid 1px black;

                 margin-block: 25px 50px;
```

```
                padding-inline-end: 70px;
                font-size: 2rem;
            }
        </style>
    </head>
    <body>
        <div >Section one</div>
        <div class="div2">Section two</div>
        <div>Section three</div>
    </body>
</html>
```

Border property :-

The border-style property sets the style of an element's four borders. This property can have from one to four values.

Examples:

- border-style: dotted solid double dashed;
    - top border is dotted
    - right border is solid
    - bottom border is double
    - left border is dashed

- border-style: dotted solid double;
    - top border is dotted
    - right and left borders are solid
    - bottom border is double

- border-style: dotted solid;
    - top and bottom borders are dotted
    - right and left borders are solid

- border-style: dotted;
    - all four borders are dotted

The CSS border properties allow you to specify the style, width, and color of an element's border.

border style :-

The border-style property specifies what kind of border to display.

The following values are allowed:

- dotted - Defines a dotted border
- dashed - Defines a dashed border
- solid - Defines a solid border
- double - Defines a double border
- groove - Defines a 3D grooved border. The effect depends on the border-color value
- ridge - Defines a 3D ridged border. The effect depends on the border-color value

- inset - Defines a 3D inset border. The effect depends on the border-color value
- outset - Defines a 3D outset border. The effect depends on the border-color value
- none - Defines no border
- hidden - Defines a hidden border

The border-style property can have from one to four values (for the top border, right border, bottom border, and the left border).

border-width:-

The border-width property specifies the width of the four borders.

The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three predefined values: thin, medium, or thick

border-color :-

The border-color property is used to set the color of the four borders.

The color can be set by:

- name - specify a color name, like "red"
- HEX - specify a HEX value, like "#ff0000"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- HSL - specify a HSL value, like "hsl(0, 100%, 50%)"
- transparent

Note: If border-color is not set, it inherits the color of the element.

Background Property :-

The background property is a shorthand property for:

- background-color
- background-image
- background-position
- background-attachment
- background-repeat
- background-clip
- background-size
- background-origin

Ex:-

```
<!D<!DOCTYPE html>

<html>

<head>

<style>

body {


  background: lightblue url("image.png") no-repeat fixed  center  ;

}

</style>

</head>
```

```
<body>

<h1>The background Property</h1>

<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
<p>This is some text</p>
```

```
<p>This is some text</p>

<p>This is some text</p>

<p>This is some text</p>

<p>This is some text</p>

<p>This is some text</p>

<p>This is some text</p>



</body>

</html>
```

Try also with other examples for this topic

Display Property :-

The display property specifies the display behavior (the type of rendering box) of an element.

The following are the short hand properties for display :-

- display-inline:- displays an element as inline element
- display-block:- displays am element as block element
- display-inline-block:- it considers element as inline element
- display-flex:- displays element as block level flex container
- display-grid :- displayes element as block - level grid container
- display-inline-flex:- displays an element as inline-flex level container
- display-inline-grid :- displays an element as inline-grid level container
- display-list-item:- it let's the element behave as list item

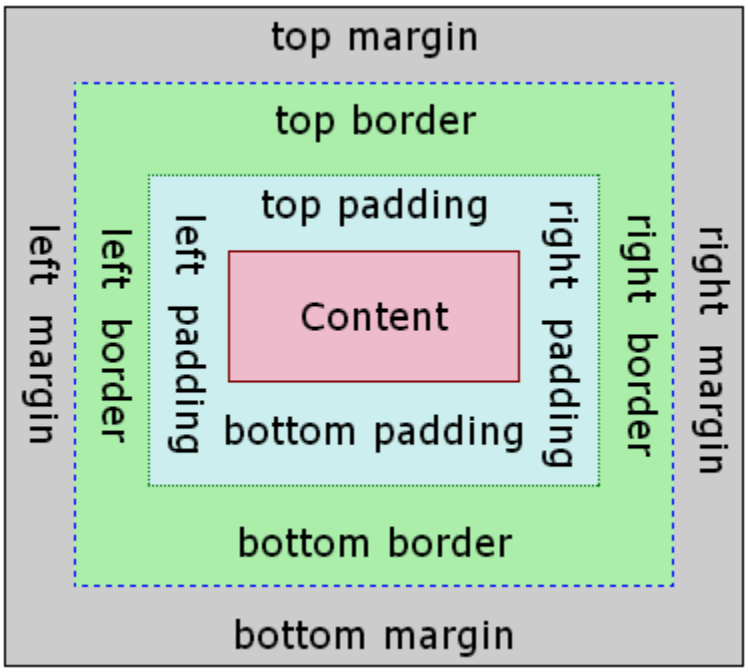
- display-run-in :- displays an element as either block or inline element
- display-inline-table:- displays element as inline level table

Try example with all the properties and apply inline table with table example

Box Model :-

In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: content, padding, borders and margins. The image below illustrates the box model



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.

Example of box model (demonstration)

```
<!DOCTYPE html>

<html>

<head>

<style>

div {

  background-color: lightgrey;

  width: 300px;

  border: 15px solid green;

  padding: 50px;

  margin: 20px;

}

</style>
```

```
</head>

<body>


<h2>Demonstrating the Box Model</h2>


<p>The CSS box model is essentially a box that wraps around every HTML element. It consists of: borders, padding, margins, and the actual content.</p>


<div>This text is the content of the box. We have added a 50px padding, 20px margin and a 15px green border. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</div>


</body>

</html>
```

Setting Width and Height of Element :-

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

When you set the width and height properties of an element with CSS, you just set the width and height of the content area. To calculate the total width and height of an element, you must also include the padding and borders.

Here is the calculation:

320px (width of content area)

+ 20px (left padding + right padding)

+ 10px (left border + right border)

= 350px (total width)

50px (height of content area)

+ 20px (top padding + bottom padding)

+ 10px (top border + bottom border)

= 80px (total height)

The total width of an element should be calculated like this:

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

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border

Css Float Property :-

The float property specifies whether an element should float to the left, right, or not at all.

Note: Absolutely positioned elements ignore the float property

```
<!DOCTYPE html>

<html>

<head>

<style>

img {

  float: right;

}

</style>

</head>

<body>

<h1>The float Property</h1>


<p>In this example, the image will float to the right in the text, and the text in the paragraph will wrap around the image.</p>


<p><img src="pineapple.jpg" alt="Pineapple"
style="width:170px;height:170px;margin-left:15px;">
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui eget tellus gravida venenatis. Integer fringilla congue eros non fermentum. Sed dapibus pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.</p>

</body>

</html>

Justify content property :-

The justify-content property aligns the flexible container's items when the items do not use all available space on the main-axis (horizontally).

Tip: Use the align-items property to align the items vertically

The `***justify-items***` property in CSS is important for aligning items within a grid container along the inline (row) axis. It allows you to control the alignment of grid items in a grid container when they do not explicitly position themselves using grid-area or similar properties.

Similar to `***align-items***` ***property*** for flex containers, ***justify-items*** enables you to align grid items horizontally within their grid areas.

CSS Outline property :-

An outline is a line drawn outside the element's border.

An outline is a line that is drawn around elements, OUTSIDE the borders, to make the element "stand out".

CSS has the following outline properties:

- outline-style
- outline-color
- outline-width
- outline-offset
- outline

Note :-

Note: Outline differs from borders! Unlike borders, the outline is drawn outside the element's border, and may overlap other content. Also, the outline is NOT a part of the element's dimensions; the element's total width and height is not affected by the width of the outline

CSS Outline styles :-

The outline-style property specifies the style of the outline, and can have one of the following values:

- dotted - Defines a dotted outline
- dashed - Defines a dashed outline
- solid - Defines a solid outline
- double - Defines a double outline
- groove - Defines a 3D grooved outline
- ridge - Defines a 3D ridged outline
- inset - Defines a 3D inset outline
- outset - Defines a 3D outset outline
- none - Defines no outline
- hidden - Defines a hidden outline

Ex:-

```
<html>

    <head>

        <style>

            .dotted{

                outline: 5px double blue;

                color: green;

                text-align: center;

            }

            p{

                outline-style: groove;

                border-style: ridge;
```

```
            }
        </style>
    </head>
    <body>
        <h1>Outline property</h1>
        <p class="dotted">Computer Science and Engineering</p>



        <p >Computer Science and Engineering</p>


    </body>
</html>
```

Try with all properties

Overflow Property :-

The overflow property specifies what should happen if content overflows an element's box.

This property specifies whether to clip content or to add scrollbars when an element's content is too big to fit in a specified area.

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

Ex:-

```
<html>
    <head>
```

```
<style>
    div.ex1{
        background-color: aqua;
        width: 110px;
        height: 110px;
        overflow: scroll;
        margin-left: 50px;
    }
    div.ex2{
        background-color:red;
        width: 110px;
        height: 110px;
        overflow:hidden ;
        margin-left: 50px;
    }
    div.ex3{
        background-color:greenyellow;
        width: 110px;
        height: 110px;
        overflow:auto;
        margin-left: 50px;
    }
```

```
            /* default it is visible */
            div.ex4{
                background-color:greenyellow;
                width: 110px;
                height: 110px;
                overflow:visible;
                margin-left: 50px;
            }




        </style>
    </head>
    <body>
        <h1>Overflow Property</h1>
        <p>Overflow property specifies whether to clip the content or to
add scrollbars , when an elements content
            is too big to fit in a specified container
        </p>

        <h2>overflow : scroll</h2>
```

```
        <div class="ex1">

            Overflow property specifies whether to clip the content or to
add scrollbars , when an elements content

            is too big to fit in a specified container

        </div>



        <h2>overflow : hidden</h2>

        <div class="ex2">

            Overflow property specifies whether to clip the content or to
add scrollbars , when an elements content

            is too big to fit in a specified container



        </div>

        <h2>overflow : auto</h2>

        <div class="ex3">

            Overflow property specifies whether to clip the content or to
add scrollbars , when an elements content

            is too big to fit in a specified container

        </div>



        <h2>overflow : clip</h2>

        <div class="ex4">
```

```
            Overflow property specifies whether to clip the content or to
add scrollbars , when an elements content

            is too big to fit in a specified container

        </div>

    </body>

</html>
```

Css Navigation Bar :-

A navigation bar needs standard HTML as a base.

In our examples we will build the navigation bar from a standard HTML list.

A navigation bar is basically a list of links, so using the <ul> and <li> elements makes perfect sense

Ex:- Vertical Navigation bar

```
<html>

    <head>

        <style>

            ul{

                list-style-type: none;

                margin: 0;

                padding: 0;

                width: 200px;
```

```
        }
        li a {
            display: block;
            width: 60px;
            padding: 8px 16px;
            background-color:beige;
            text-decoration: none;
        }
        li a:hover {
            background-color: black;
            color: aliceblue;
        }

    </style>
</head>
<body>
    <ul>
        <li><a href="#home">Home</a></li>
        <li><a href="#news">News</a></li>
        <li><a href="#contact">Contact</a></li>
        <li><a href="#about">About</a></li>
    </ul>
```

```
    </body>
</html>
```

Css navigation Bar (horizontal)

There are two ways to create a horizontal navigation bar. Using inline or floating list items.One way to build a horizontal navigation bar is to specify the <li> elements as inline.

Ex:-

```
<html>
    <head>
        <style>
            body{
                overflow: scroll;
            }
            ul{
                list-style-type: none;
                margin: 0;
                padding: 0;
                overflow: hidden;
                background-color: aqua;
                border: 1px solid #e7e7e7;
            }
```

```
li a{
    text-decoration: none;
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    float: left;
    position: sticky;
    position: -webkit-sticky;
    top: 0;

}
/* li a:hover{
    background-color:black;
} */
/* li a:hover:not(.active) {
background-color: #ddd;
 }
li a.active {
    color: white;
    background-color: #04AA6D;
}
```

```
 */


li a:hover {
  background-color: #111;
}


/* .active {
  background-color: #4CAF50;
} */
        </style>
    </head>
    <body>
        <ul>
            <li><a  href="#home">Home</a></li>
            <li><a href="#news">News</a></li>
            <li><a href="#contact">Contact</a></li>
            <li><a href="#about">About</a></li>
        </ul>
    </body>
</html>
```

CSS Scroll Property :-

This property is used for smooth animation of scroll position instead of a scroll jump. When the user clicks on links it smoothly performs its operation. It is used to visit one link to another link within a scrollable box.

Default Value: auto

Property:

- **smooth:** This property is used to specify the animation effect of the scroll between the elements within the scrollable box.
- **auto**: It is used to specify the straight jump scroll effect visit to one link to another link within a scrolling box.

Ex:-

```
<html>

    <head>

        <style>

            html {

                scroll-behavior: smooth;

            }



            #section1 {
```

```
            height: 800px;
            background-color: bisque;
        }
        #section2 {
            height: 600px;
            background-color:yellow;
        }

    </style>
</head>
<body>
    <div class="main" id="section1">
        <h2>Section 1</h2>
        <p> Click on the link to see the "smooth" scrolling effect</p>
        <a href="#section2">Click me to have smooth scroll to section
2 </a>

        <div class="main" id="section2">
            <h2>Section 2</h2>
```

```
                <a href="#section1">Click me to have smooth scroll to
section 1</a>



            </div>

        </div>

    </body>

</html>
```

Scroll-margin property :-

The scroll-margin property specifies the distance between the snap position and the container.

This means that when you stop scrolling, the scrolling will quickly adjust and stop at a specified distance between the snap position and the container.

The scroll-margin property is a shorthand property for the following properties:

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

Values for the scroll-margin property can be set in different ways:

If the scroll-margin property has four values:

- scroll-margin: 15px 30px 60px 90px;
    - top distance is 15px
    - right distance is 30px
    - bottom distance is 60px

    - left distance is 90px

If the scroll-margin property has three values:

- scroll-margin: 15px 30px 60px;
    - top distance is 15px
    - left and right distances are 30px
    - bottom distance is 60px

If the scroll-margin property has two values:

- scroll-margin: 15px 30px;
    - top and bottom distances are 15px
    - left and right distances are 30px

If the scroll-margin property has one value:

- scroll-margin: 10px;
    - all four distances are 10px

Ex :-

```
<html>

    <head>

        <style>

            .page{

                width: 270px;

                height: 278px;

                color: aliceblue;

                font-size: 50px;
```

```
            display: flex;
            align-items: center;
            justify-content: center;

        }
        .container{
            width: 300px;
            height: 300px;
            overflow-x: hidden;
            overflow-y: auto;

        }

    </style>
</head>
<body>
    <div class="container">
        <div class="page" style="background-color: aquamarine;
scroll-margin-top: 0px;">
            Scroll-margin
```

```
            </div>

            <div class="page" style="background-color:blueviolet;
scroll-margin-top: 20px;">

                Scroll-margin

            </div>

            <div class="page" style="background-color:brown;
scroll-margin-top: 40px;">

                Scroll-margin

            </div>

            <div class="page" style="background-color:blanchedalmond;
scroll-margin-top: 50px;">

                Scroll-margin

            </div>



        </div>

    </body>

</html>
```

CSS scroll-margin-block :-

The scroll-margin-block property specifies the distance in block direction, between the snap position and the container.

This means that when you stop scrolling, the scrolling will quickly adjust and stop at a specified distance in block direction, between the snap position and the container.

Values for the scroll-margin-block property can be set in different ways:

If the scroll-margin-block property has two values:

- scroll-margin-block: 10px 50px;
    - distance at start is 10px
    - distance at end is 50px

If the scroll-margin-block property has one value:

- scroll-margin-block: 10px;
    - distance at start and end is 10px

To see the effect from the scroll-margin-block property, the scroll-margin-block and scroll-snap-align properties must be set on the child elements, and the scroll-snap-type property must be set on the parent element.

Ex:-

CSS scroll-margin-inline:-

The scroll-margin-inline property specifies the distance in the inline direction, between the snap position and the container.

This means that when you stop scrolling, the scrolling will quickly adjust and stop at a specified distance in the inline direction, between the snap position and the container.

If the scroll-margin-inline property has two values:

- scroll-margin-inline: 20px 70px;
    - distance at start is 20px

    - distance at end is 70px

If the scroll-margin-inline property has one value:

- scroll-margin-inline: 20px;
    - distance at start and end is 20px

Ex:-

```
<html>
    <head>
        <style>
            .page{
                width: 278px;
                height: 296px;
                color: azure;
                font-size: 60px;
                display: flex;
                justify-content: center;
                align-items: center;
            }
            .container{
                width: 300px;
                height: 300px;
                overflow-x: hidden;
                overflow-y: auto;
```

```
            }



        </style>
    </head>
    <body>
        <div class="container">
            <div class="page" style="background-color: aqua;
scroll-margin-block: 90px;">
                Scroll margin block- 1
            </div>
            <div class="page" style="background-color:red;
scroll-margin-block: 90px;">
                Scroll margin block - 2
            </div>
            <div class="page" style="background-color:yellow;
scroll-margin-block: 90px;">
                Scroll margin block - 3
            </div>
            <div class="page" style="background-color:black;
scroll-margin-block: 90px;">
                Scroll margin block - 4
            </div>
        </div>
    </body>
</html>
```

Same example for block and inline just change the properties

JavaScript

Introduction :-

JavaScript is a *lightweight, cross-platform*, *single-threaded,* and *interpreted compiled* programming language. It is also known as the scripting language for web pages. It is well-known for the development of web pages, and many non-browser environments also use it.

JavaScript can be used for Client-side developments as well as Server-side developments. JavaScript is both an imperative and declarative type of language. JavaScript contains a standard library of objects, like Array , date and Math, and a core set of language elements like operators , control structures, and statements.

JavaScript is a high-level, interpreted programming language primarily used for making web pages interactive. It allows you to implement complex features on web pages, such as handling user interactions, manipulating the DOM (Document Object Model), and making asynchronous requests to servers.

Features :-

Client-Side Scripting: JavaScript code runs on the client-side (in the user's browser), allowing for dynamic content and interaction with the user interface. This enables web developers to create responsive and interactive websites.

Cross-platform Compatibility: JavaScript is supported by all major web browsers (Chrome, Firefox, Safari, Edge, etc.), making it a reliable choice for web development. It is also increasingly used beyond web browsers (e.g., server-side with Node.js, desktop applications with Electron.js).

Simple and Easy to Learn: JavaScript has a syntax similar to other programming languages like C and Java, making it relatively easy to pick up for developers familiar with these languages. Its dynamic typing and flexible syntax reduce the overhead of learning complex type systems.

Versatility: Apart from web development, JavaScript can be used for a variety of applications, including mobile app development (using frameworks like React Native), server-side scripting (with Node.js), and even for creating desktop applications.

Event-Driven and Asynchronous: JavaScript is inherently event-driven, meaning it can respond to user actions such as clicks, scrolls, and keyboard inputs. It also supports asynchronous programming, allowing tasks to run in the background without blocking the main program execution. This is crucial for handling tasks like fetching data from servers without freezing the user interface.

Rich Interfaces with DOM Manipulation: JavaScript allows developers to manipulate the Document Object Model (DOM) of a webpage, enabling dynamic changes to the content and structure of the page in response to user actions or other events.

Support for Libraries and Frameworks: JavaScript has a vast ecosystem of libraries and frameworks (e.g., React, Angular, Vue.js) that simplify complex tasks and provide pre-built solutions for common web development challenges. These tools enhance productivity and maintainability of JavaScript codebases.

Community and Resources: JavaScript benefits from a large and active community of developers, which contributes to the availability of tutorials, documentation, and open-source projects. This community support is invaluable for learning and troubleshooting JavaScript-related issues.

Security: While JavaScript runs on the client-side and can be manipulated by users, modern browsers have security measures in place to prevent malicious actions (like cross-site scripting attacks). Developers must follow best practices to ensure the security of their JavaScript applications.

Continuous Evolution: JavaScript evolves rapidly with new language features and updates (e.g., ECMAScript standards), ensuring that developers have access to modern programming capabilities and performance improvements.

Pros :-

Versatility: JavaScript can be used for both front-end (client-side) and back-end (server-side) development, thanks to frameworks like Node.js. This versatility allows developers to use a single language across different parts of their applications.

Ease of Learning: JavaScript has a relatively forgiving syntax compared to other programming languages, making it accessible for beginners. It doesn't require compilation and can be directly run in any web browser, simplifying the development and testing process.

Interactivity: JavaScript enables interactive web pages by allowing developers to respond to user actions in real-time without reloading the entire page. This improves user experience by creating dynamic and responsive interfaces.

Rich Interfaces: With JavaScript, developers can manipulate the DOM (Document Object Model) to dynamically change the content and styling of web pages. This ability to update parts of a page without full refreshes leads to smoother and more interactive user interfaces.

Large Ecosystem: JavaScript has a vast ecosystem of libraries (like jQuery, React, Angular, Vue.js) and frameworks that simplify complex tasks and

accelerate development. These tools provide pre-built components, extensive documentation, and strong community support.

Asynchronous Programming: JavaScript supports asynchronous programming, which allows tasks to run concurrently without blocking the main thread. This is crucial for handling operations like fetching data from servers or performing animations without freezing the user interface.

Performance: Modern JavaScript engines (like V8 in Chrome and Node.js) have significantly improved performance, making JavaScript a viable option for high-performance applications. Frameworks and tools built on JavaScript also optimize performance through efficient rendering and data handling.

Cross-platform Compatibility: JavaScript is supported by all major web browsers and can run on multiple platforms, including desktops, mobile devices, and servers. This cross-platform compatibility ensures consistent behavior across different environments.

Community and Support: JavaScript has a large and active community of developers who contribute to open-source projects, share knowledge through forums and tutorials, and provide assistance on platforms like Stack Overflow. This community support helps developers solve problems quickly and stay updated with best practices.

Continuous Improvement: JavaScript continues to evolve with regular updates to the ECMAScript standard (the specification that JavaScript follows). New features and improvements enhance language capabilities, improve performance, and address developer needs.

Cons:-

Browser Compatibility: Different web browsers may interpret JavaScript code differently, leading to inconsistencies in behavior and functionality across

platforms. Developers often need to write additional code or use polyfills to ensure compatibility with older browsers.

Security: Since JavaScript code runs on the client-side, it is inherently vulnerable to attacks like cross-site scripting (XSS). Developers need to implement security best practices, such as input validation and escaping output, to prevent malicious code execution.

Performance: While modern JavaScript engines have improved performance, JavaScript can still be slower compared to lower-level languages like C++ or Java. Intensive computations or complex applications may require careful optimization to maintain responsiveness.

Single-threaded Execution: JavaScript is single-threaded, meaning it can only execute one operation at a time on a single thread of execution. This can lead to blocking behavior if intensive operations are not managed properly, affecting responsiveness in some scenarios.

Lack of Static Typing: JavaScript is dynamically typed, which means variable types are determined at runtime rather than compile-time. This can lead to errors that might only surface during runtime, making it harder to catch bugs early in development.

Callback Hell: Asynchronous programming in JavaScript often relies on callbacks or nested callbacks (known as callback hell), which can make code difficult to read, maintain, and debug. Promises and async/await were introduced to alleviate this issue, but it still requires careful handling.

Scalability: Large-scale JavaScript applications can become difficult to manage and scale due to the language's flexibility and lack of strict structure. Developers may need to rely on design patterns, modularization, and frameworks to maintain code organization and scalability.

Tooling and Dependency Management: JavaScript's ecosystem, while extensive, can also be fragmented with a wide array of libraries, frameworks, and tools. Managing dependencies, ensuring compatibility, and staying up-to-date with the latest versions can be challenging.

Learning Curve: While JavaScript's syntax is relatively simple, mastering advanced concepts and best practices can take time and effort. Beginners may encounter challenges understanding concepts like closures, prototypal inheritance, and functional programming paradigms.

Evolution and Compatibility: JavaScript evolves rapidly with new features and updates to the ECMAScript standard. While this brings improvements, it can also introduce compatibility issues with older codebases or browsers that lag behind in implementing new features.

JavaScript Code Execution Phase :-

JavaScript is a *synchronous* (Moves to the next line only when the execution of the current line is completed) and *single-threaded* (Executes one command at a time in a specific order one after another serially) language. To know behind the scenes of how JavaScript code gets executed internally, we have to know something called **Execution Context** and its role in the execution of JavaScript code.

Execution Context: Everything in JavaScript is wrapped inside Execution Context, which is an abstract concept (can be treated as a container) that holds the whole information about the environment within which the current JavaScript code is being executed.

Now, an Execution Context has two components and JavaScript code gets executed in two phases.

- **Memory Allocation Phase:** In this phase, all the functions and variables of the JavaScript code get stored as a key-value pair inside

the memory component of the execution context. In the case of a function, JavaScript copied the whole function into the memory block but in the case of variables, it assigns *undefined* as a placeholder.

- **Code Execution Phase:** In this phase, the JavaScript code is executed one line at a time inside the Code Component (also known as the Thread of execution) of Execution Context.

Let's see the whole process through an example.

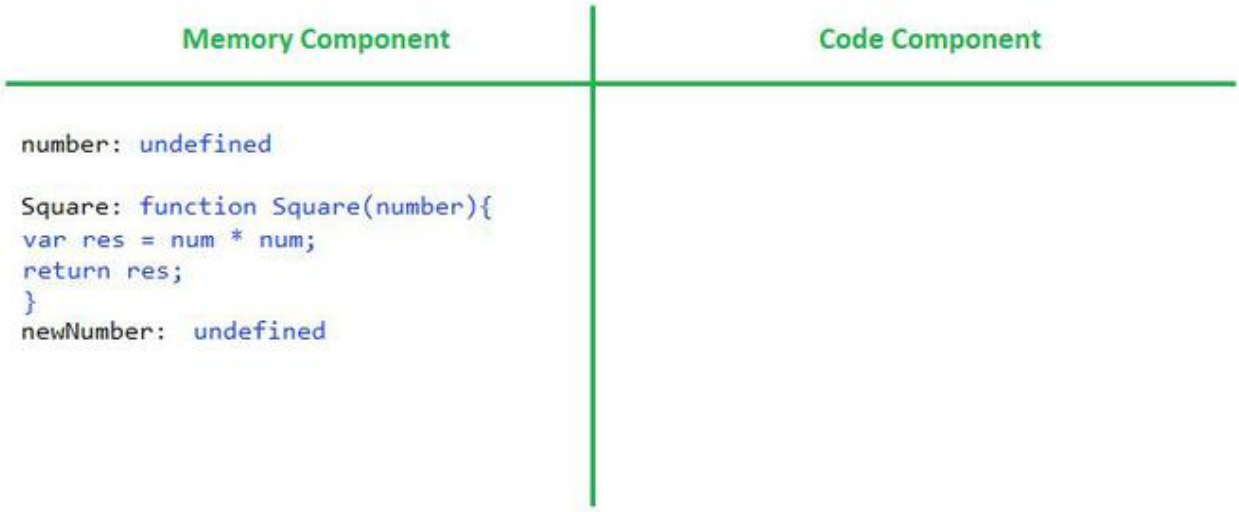
Ex:-

```
var number = 2;

function Square (n) {

    var res = n * n;

    return res;

}

var newNumber = Square(3);
```

Explanation :-

In the above JavaScript code, there are two variables named *number* and *newNumber* and one function named *Square* which is returning the square of the number. So when we run this program, Global Execution Context is created.

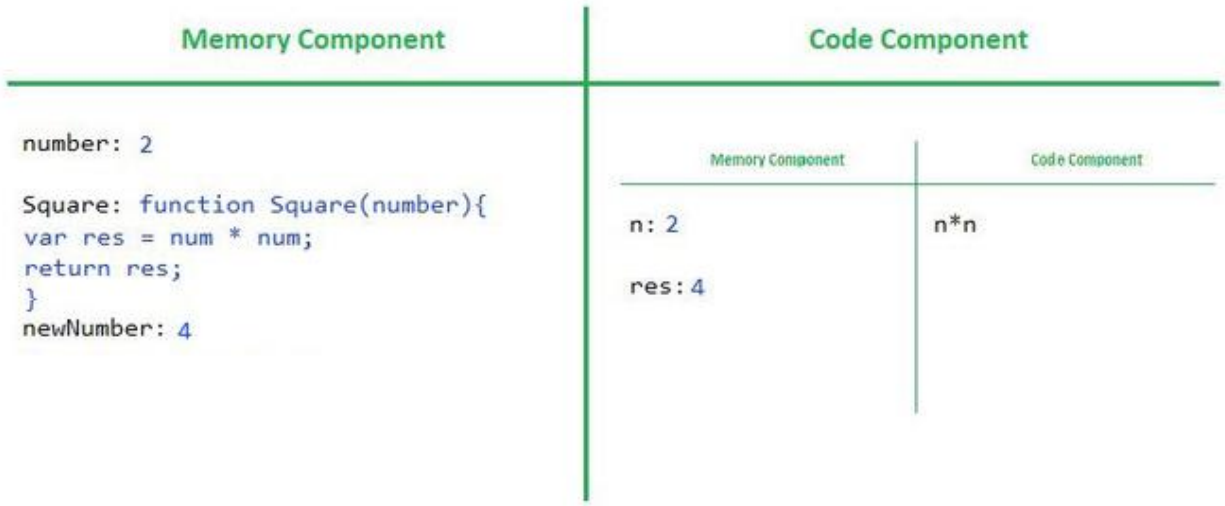
So, in the Memory Allocation phase, the memory will be allocated for these variables and functions like this.



In the Code Execution Phase, JavaScript being a single thread language again runs through the code line by line and updates the values of function and variables which are stored in the Memory Allocation Phase in the Memory Component.

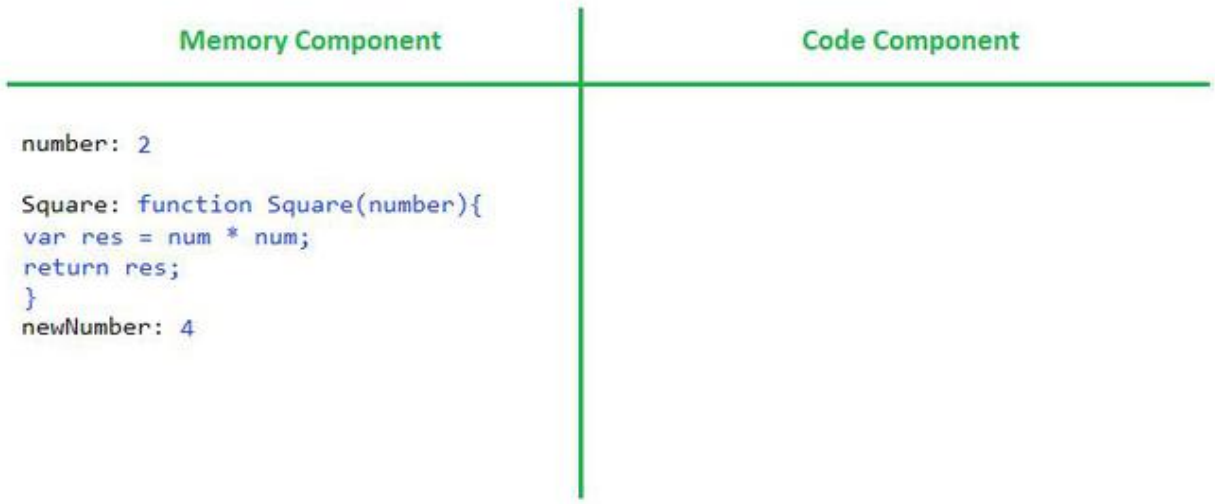
So in the code execution phase, whenever a new function is called, a new Execution Context is created. So, every time a function is invoked in the Code Component, a new Execution Context is created inside the previous global execution context.

So again, before the memory allocation is completed in the Memory Component of the new Execution Context. Then, in the Code Execution Phase of the newly created Execution Context, the global Execution Context will look like the following.



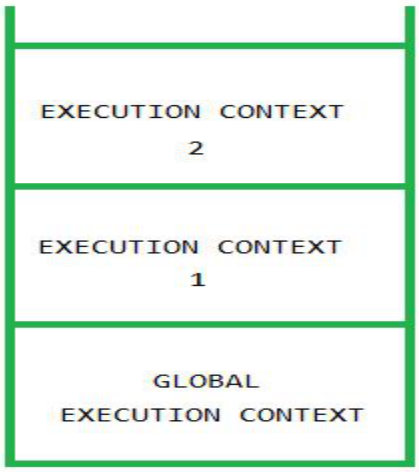
As we can see, the values are assigned in the memory component after executing the code line by line, i.e. *number: 2, res: 4, newNumber: 4*.

After the *return* statement of the invoked function, the returned value is assigned in place of undefined in the memory allocation of the previous execution context. After returning the value, the new execution context (temporary) gets completely deleted. Whenever the execution encounters the return statement, It gives the control back to the execution context where the function was invoked.



After executing the first function call when we call the function again, JavaScript creates another temporary context where the same procedure repeats accordingly (memory execution and code execution). In the end, the global execution context gets deleted just like child execution contexts. The whole execution context for the instance of that function will be deleted

Call Stack: When a program starts execution JavaScript pushes the whole program as global context into a stack which is known as **Call Stack** and continues execution. Whenever JavaScript executes a new context and just follows the same process and pushes to the stack. When the context finishes, JavaScript just pops the top of the stack accordingly.



When JavaScript completes the execution of the entire code, the Global Execution Context gets deleted and popped out from the Call Stack making the Call stack empty.

Types of Execution Context :-

There are two types of execution context .

- **Global Execution Context :-** Whenever Javascript engine (v8 engine) receives a script file , it first creates a default execution context called as “ Global execution Context “ .This is the base and the default execution context where all the code that is not inside the function gets executed . For every Js file there will be only one GEC .
- **Function Execution Context :-** Whenever a function is called , the JS engine creates a different type of execution context called “Function Execution Context “ within the GEC to evaluate and execute the code within that function . Since every function call gets its own FEC , there can be more than one FEC in the runtime of a script .

DOM (Document Object Model)

Html DOM :- HTML DOM is hierarchical representation of HTML documents.

It defines the structure and properties of elements on a webpage, enabling JavaScript to dynamically access, manipulate, and update content, enhancing interactivity and functionality.

It is called a Logical structure because DOM doesn't specify any relationship between objects.

Why is DOM Required?

HTML is used to **structure** the web pages and **Javascript** is used to add **behavior** to our web pages. When an HTML file is loaded into the browser, the JavaScript can not understand the HTML document directly. So it interprets and interacts with the Document Object Model (DOM), which is created by the browser based on the HTML document.

DOM is basically the representation of the same HTML document but in a tree-like structure composed of objects. JavaScript can not understand the tags(<h1>H</h1>) in HTML documents but can understand object h1 in DOM.

JavaScript interprets DOM easily, using it as a bridge to access and manipulate the elements. DOM Javascript allows access to each of the objects (h1, p, etc) by using different functions.

The Document Object Model (DOM) is essential in web development for several reasons:

- **Dynamic Web Pages:** It allows you to create dynamic web pages. It enables JavaScript to access and manipulate page content, structure, and style dynamically which gives interactive and responsive web experiences, such as updating content without reloading the entire page or responding to user actions instantly.
- **Interactivity:** With the DOM, you can respond to user actions (like clicks, inputs, or scrolls) and modify the web page accordingly.
- **Content Updates:** When you want to update the content without refreshing the entire page, the DOM enables targeted changes making the web applications more efficient and user-friendly.
- **Cross-Browser Compatibility:** Different browsers may render HTML and CSS in different ways. The DOM provides a standardized way to interact with page elements.
- **Single-Page Applications (SPAs):** Applications built with frameworks such as React or Angular, heavily rely on the DOM for efficient rendering and updating of content within a single HTML page without reloading the full page.

Structure of DOM

DOM can be thought of as a Tree or Forest (more than one tree). The term **structure model** is sometimes used to describe the tree-like representation of a document.

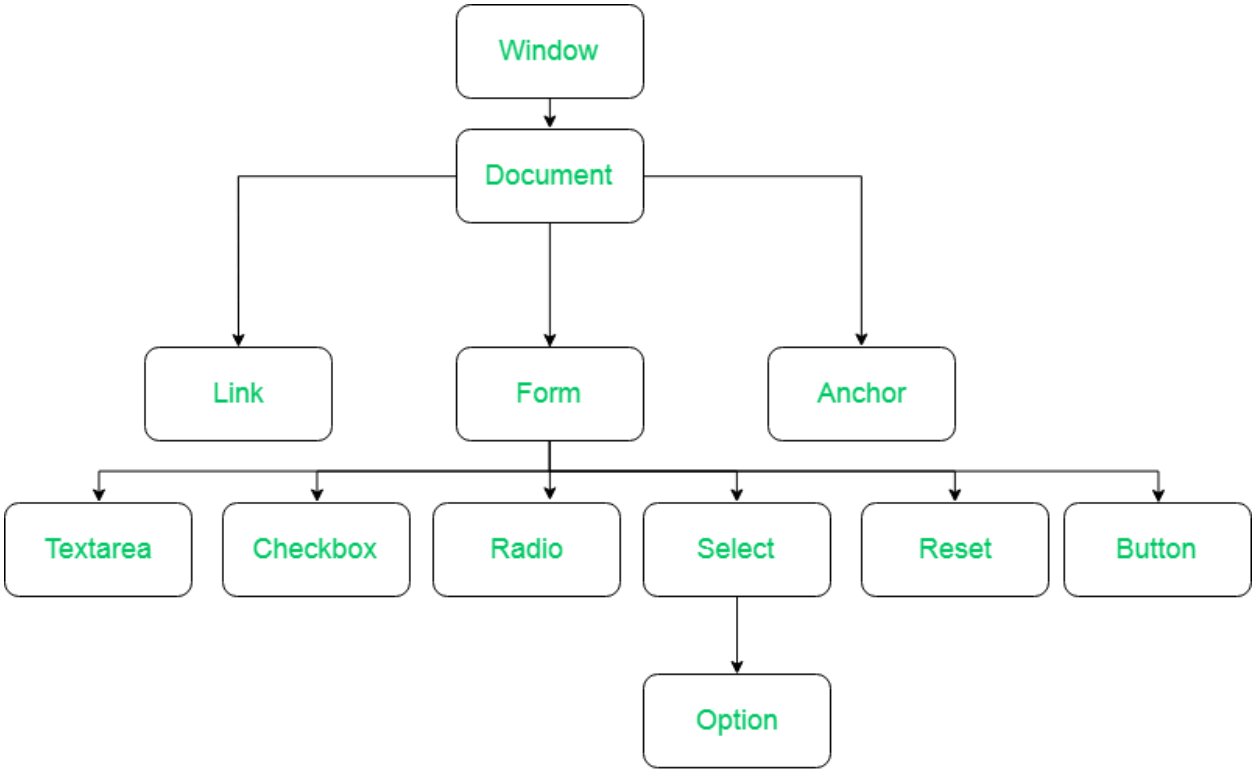
Each branch of the tree ends in a node, and each node contains objects Event listeners can be added to nodes and triggered on an occurrence of a given event. One important property of DOM structure models is ***structural isomorphism***: if any two DOM implementations are used to create a representation of the same document, they will create the same structure model, with precisely the same objects and relationships.

Why is DOM called an Object Model?

Documents are modeled using objects, and the model includes not only the structure of a document but also the behavior of a document and the objects of which it is composed like tag elements with attributes in HTML.

Properties of DOM

Let’s see the properties of the document object that can be accessed and modified by the document object.



- **Window Object:** Window Object is object of the browser which is always at top of the hierarchy. It is like an API that is used to set and access all the properties and methods of the browser. It is automatically created by the browser.
- **Document object:** When an HTML document is loaded into a window, it becomes a document object. The ‘document’ object has various properties that refer to other objects which allow access to and modification of the content of the web page. If there is a need to access any element in an HTML page, we always start with accessing the ‘document’ object. Document object is property of window object.

- **Form Object:** It is represented by ***form*** tags.
- **Link Object:** It is represented by ***link*** tags.
- **Anchor object:** It is represented by ***a href*** tag.
- **Form Control Elements:** Form can have many control elements such as text fields, buttons, radio buttons, checkboxes, etc.

DOM Methods :-

Ex:-

```
<html>
    <input type="text" id="username">
    <button name="button" onclick="sample()">Capture</button>
    <p> This is paragraph</p>
    <script>
        function sample(){
         var username= document.getElementById("username");
         var p= document.getElementsByTagName("p");
         console.log(p[0]);

         p[0].style.color="red";
         p[0].style.fontSize="50px";
```

```
}
var button=document.getElementsByName("button");
console.log(button[0]);
button[0].onclick=sample;
    </script>
</html>
```

Variables :-

Variables are used to store the data , the values that we store on variables are reusable.We can allocate the values to the variable using assignment operator (=) .

Ex :- age = 20;

Here age is an identifier, it is assigned with a value of 20 . Identifier assigned with a value is variable.

Identifier :-

JavaScript variables must have unique names; these names are called identifiers.

Basic rules to declare a variable in JavaScript are :-

- These are case-sensitive
- Can only begin with a letter, underscore("_") or "$" symbol
- It can contain letters, numbers, underscore, or "$" symbol
- A variable name cannot be a reserved keyword.

JavaScript is a dynamically typed language so the type of variables is decided at runtime. Therefore there is no need to explicitly define the type of a variable. We can declare variables in JavaScript in three ways:

var, let const

Var :-

The **JavaScript var statement** declares variables with function scope or globally. Before ES6, **var** was the sole keyword for variable declaration, without block scope, unlike **let** and **const**.

Function Scope :-

The variables declared inside a function are function-scoped and cannot be accessed outside the function.The variables declared using the var statement are hoisted at the top and are initialized before the execution of code with a default value of undefined. The variables declared in the global scope that is outside any function cannot be deleted.

Ex :-

```
var test = 12

function foo(){

    console.log(test);

}

foo();
```

Declare multiple variables in a single statement.

```
var test1 = 12,

    test2= 14,

    test3 = 16

function foo(){

    console.log(test1, test2, test3);

}

foo();
```

Next we will see variable hoisting example

Ex :-

```
console.log(test);

var test = 12;
```

Explanation: We get the output without any error because the variable test is hoisted at the top even before the execution of the program begins and the variable is initialized with a default value of undefined.

Ex:- 2

Redeclaring the variable in same global block

```
 var test1=30;

 var test1=100;

console.log(test1);
```

We didn't get any error while redeclaring the same variable , if we did this with the let keyword it would throw an error .

Ex:- 3 (We will redeclare the variable in another scope and see how it is the original variable.)

```
var age = 12;

function foo(){

    var age = 100;

    console.log(age);

}
```

```
foo();

console.log(age);
```

Explanation: We did not get any error while redeclaring the variable inside another function scope and the original value of the variable is preserved.

Ex:- 4

In this example, we will try to delete a global variable declared using var in the ‘use strict’ mode

```
 'use strict';

var test = 12;

delete(test);

console.log(test);
```

Explanation: Whenever a variable is declared using var in global scope it cannot be configured. Hence it cannot be deleted using the delete keyword. and an error is thrown.

When to Use var, let, or const

- We declare variables using const if the value should not be changed
- We use const if the type of the variables should not be changed such as working with Arrays and objects
- We should use let if we want mutable value or we can not use const
- We use var only if we support old browsers

let :-

The let keyword in JavaScript is used to make variables that are scoped to the block they're declared in. Once you've used let to define a variable, you cannot declare it again within the same block. It's important to declare *let variables before using them*.

Block Scope :-

The variables which are declared inside the { } block are known as block-scoped variables. variables declared by the var keyword cannot be block-scoped.

Ex :- 1 (In this example, the num variable is block-scoped and it cannot be accessed outside the block. If we try to access the variable outside the block it throws a reference error.)

```
{
    let num=20;
    console.log(num);
}
console.log(num); // num is not defined error
```

Global Scope :-

A global scope variable is a variable declared in the main body of the source code, outside all functions.

Ex:-2 (In this example, the num variable is a globally scoped variable and it can be accessed from anywhere in the program.)

```
let num=20;

console.log(num);

function fun1() {

    console.log(num);

}

fun1();
```

Function Scope :-

A function scope variable is a variable declared inside a function and cannot be accessed outside the function.

Ex:- 3 (In this example, the num variable is declared inside the function and cannot be accessed outside the function.)

```
let num=20;

console.log(num);

function fun1() {

    let num2=30;

    console.log(num);

    console.log(num2);

}

fun1();

console.log(num2);
```

Properties of let :-

1. **Redeclaring Variables in different blocks :-** The variables declared using let can be redeclared inside other blocks.

Ex:- 4 (In this example, variable x is redeclared inside other blocks.)

```
let x=40;

{

    let x=45;

    console.log(x);

}

console.log(x);
```

2. **Redeclaring the variables in the same block :-** We cannot redeclare variables using the let keyword inside the same blocks. It will throw an error.

Ex:- 5 (In this example, variable x is redeclared inside the same blocks.)

```
let x=40;

{

    let x=45;

    console.log(x);

}

console.log(x);

let x=34;

console.log(x);
```

3. **Does not support hoisting of variables :-** The behavior of moving the variable declarations on the top of script is called as hoisting .let keyword does not support hoisting .

Ex:- (.let keyword does not support hoisting)

```
console.log(x);

let x=20;
```

Const :-

The const keyword in JavaScript is used to define variables that cannot be changed once they're assigned a value. This prevents any modifications to the variable's value.

Additionally, const doesn't allow redeclaration of the same variable within the same block, and it provides block scope. It was introduced in ES2015 (ES6) for creating immutable variables.

Properties:

- Cannot be reassigned.
- It has Block Scope
- It can be assigned to the variable on the declaration line.
- It's a Primitive value.
- The property of a const object can be changed but it cannot be changed to a reference to the new object
- The values inside the const array can be changed, it can add new items to const arrays but it cannot reference a new array.
- Re-declaring of a const variable inside different block scopes is allowed.
- Cannot be Hoisted.
- Creates only read-only references to value.

Ex:- 1 (Cannot be Reassigned)

```
const x=30;
console.log(x);
x=40; // error ( Assignment to constant variable )
```

Ex:- 2 (It describes the const variable which contains the Block Scope.)

```
const x = 22;
{
    const x = 90;
    console.log(x);

    {
        const x = 77;
        console.log(x);
    }
    {
        const x = 45;
        console.log(x);
    }
}
console.log(x);
```

Ex:- 3 (Variables must be assigned)

const x;

x = 12; // error (`Uncaught SyntaxError: Missing initializer in const declaration )`

Ex:- 4 (Cannot be hoisted)

x = 3;

console.log(x);

const x; // error (`Uncaught SyntaxError: Missing initializer in const declaration )`

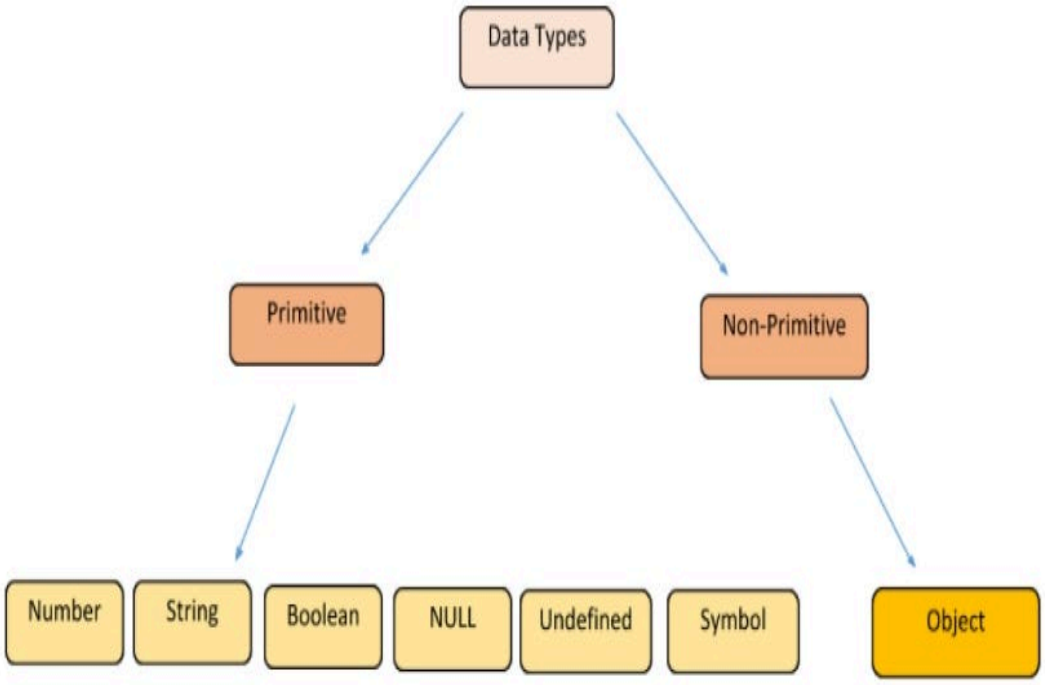
Data Types :-

The Concept of Data Types

In programming, data types are an important concept.

To be able to operate on variables, it is important to know something about the type.

JAVASCRIPT DATA TYPES



JavaScript is a **dynamically typed** (also called loosely typed) scripting language. In JavaScript, variables can receive different data types over time. The latest ECMAScript standard defines eight data types Out of which seven data types are **Primitive (predefined)** and one **complex or Non-Primitive**.

Primitive Data Types

The predefined data types provided by JavaScript language are known as primitive data types. Primitive data types are also known as in-built data types.

- **Number:** JavaScript numbers are always stored in double-precision 64-bit binary format IEEE 754. Unlike other programming languages, you don't need int, float, etc to declare different numeric values.
- **String:** JavaScript Strings are similar to sentences. They are made up of a list of characters, which is essentially just an "array of characters, like "Hello GeeksforGeeks" etc.
- **Boolean:** Represent a logical entity and can have two values: true or false.
- **Null:** This type has only one value that is *null*.
- **Undefined:** A variable that has not been assigned a value is *undefined*.
- **Symbol:** Symbols return unique identifiers that can be used to add unique property keys to an object that won't collide with keys of any other code that might add to the object.
- **BigInt:** BigInt is a built-in object in JavaScript that provides a way to represent whole numbers larger than 2^53-1.

Non-Primitive Data Types

The data types that are derived from primitive data types of the JavaScript language are known as non-primitive data types. It is also known as derived data types or reference data types.

- **Object:** It is the most important data type and forms the building blocks for modern JavaScript.
- Array :- It is datatype but special type of object
- Function:-It is considered as non primitive data type , it is a type of object and subtype of an Object

Ex :- 1

```
let x= 16 + 20; // addition

let y= 20  + "react"; // concat

// strings has highest priority

console.log(x);

console.log(y);

console.log(typeof(x));

let z= 30 * "react"; // NaN -> not a number

console.log(z);
```

Ex :- 2

```
let x= "20" + "React";

let y="String value";

console.log(x);

console.log(y);
```

```
let x=true;

console.log(typeof(x));
```

Ex:- 3

```
let x=null;

console.log(typeof(x)); // "object"
```

Ex :- 4

```
const languages = ["Java", 30 , true , "React"];

console.log(languages);

console.log(typeof(languages));



const sample = [];

sample[0] = 1;

sample[1] = 17;

sample[2] = "Sample";

sample[3] = true;



console.log(sample[5]); // undefined
```

```
console.log(typeof(sample)); // object



const example = new Array("ex1" , "ex2", 45 , true);

console.log(example);

example[2]=68;

console.log(example);

example[4]=90;

console.log(example);

example.pop();

console.log(example);
```

Arrays :-

Array is an object that can store multiple values at once. It comes under non - non-primitive data type because it contains a collection of values . It is also a data structure that allows you to store multiple heterogeneous values in a single variable.

Array is an Object subtype it inherits from Object.prototype and has additional array-specific properties and methods . This unique blend of datatype and object makes Javascript arrays powerful and flexible.

Why do we use Arrays ?

If you have list of items storing that in single variables would be as follows

let x=40;

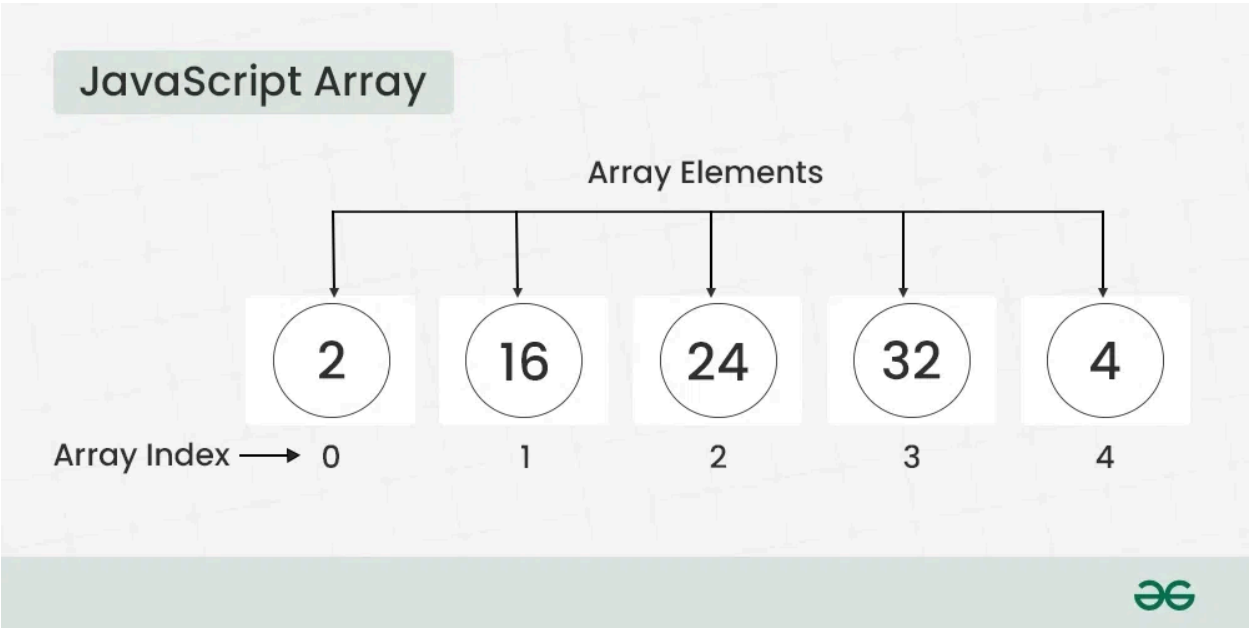
let y=70;

let h=80;

You can do the same and loop through it if you have some 5 to 10 variables but what if you have 300 or 400 ?

The solution is Array

As mentioned above it can hold multiple values in a single variable . You can access it by its index number .



Basic Terminologies of JavaScript Array

- **Array:** A data structure in JavaScript that allows you to store multiple values in a single variable.
- **Array Element:** Each value within an array is called an element. Elements are accessed by their index.
- **Array Index:** A numeric representation that indicates the position of an element in the array. JavaScript arrays are zero-indexed, meaning the first element is at index 0.
- **Array Length:** The number of elements in an array. It can be retrieved using the length property.

Declaration of an Array

There are basically two ways to declare an array i.e. Array Literal and Array Constructor.

1. Creating an Array using Array Literal

Creating an array using array literal involves using square brackets [] to define and initialize the array. This method is concise and widely preferred for its simplicity.

Syntax :- const arrone = [val1, val2 ,...];

Ex:-

```
// Creating an Empty Array
```

```
const names = [];

console.log(names);

//Creating an Array and Initializing with Values

const courses = ["HTML", "CSS", "Javascript", "React"];

console.log(courses);
```

2. Creating an Array using Array Constructor (JavaScript new Keyword)

The “Array Constructor” refers to a method of creating arrays by invoking the Array constructor function. This approach allows for dynamic initialization and can be used to create arrays with a specified length or elements.

Syntax :- `let arrayName = new Array();`

Ex:-

// Declaration of an empty array

// using Array constructor

let names = new Array();

console.log(names);

// Creating and Initializing an array with values

```
let courses = new Array("HTML", "CSS", "Javascript", "React");

console.log(courses);



// Initializing Array while declaring

let arr = new Array(3);

arr[0] = 10;

arr[1] = 20;

arr[2] = 30;

console.log(arr);
```

Ex :-

```
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
console.log(cars);
```

Ex:- (You can also create an array, and then provide the elements:)

```
const cars = [];

cars[0]= "Saab";

cars[1]= "Volvo";

cars[2]= "BMW";

console.log(cars);
```

Basic Operations on Arrays :-

1. Accessing Elements of an Array:-Any element in the array can be accessed using the index number. The index in the arrays starts with 0.

Ex :-

```
// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "Javascript", "React"];

// Accessing Array Elements
console.log(courses[0]);
console.log(courses[1]);
console.log(courses[2]);
console.log(courses[3]);
```

2. Accessing the First Element of an Array

The array indexing starts from 0, so we can access the first element of array using the index number.

Ex:-

// Creating an Array and Initializing with Values

```
let courses = ["HTML", "CSS", "JavaScript", "React"];

// Accessing First Array Elements

let firstItem = courses[0];

console.log("First Item: ", firstItem);
```

3. Accessing the Last Element of an Array

We can access the last array element using [array.length – 1] index number.

Ex :-

```
// Creating an Array and Initializing with Values

let courses = ["HTML", "CSS", "JavaScript", "React"];

// Accessing Last Array Elements

let lastItem = courses[courses.length - 1];

console.log("First Item: ", lastItem);
```

4. Modifying the Array Elements

Elements in an array can be modified by assigning a new value to their corresponding index.

Ex :-

```
// Creating an Array and Initializing with Values

let courses = ["HTML", "CSS", "Javascript", "React"];

console.log(courses);

courses[1]= "Bootstrap";
```

```
console.log(courses);
```

5. Adding Elements to the Array

Elements can be added to the array using methods like push() and unshift().

Ex :-

```
// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "Javascript", "React"];

// Add Element to the end of Array
courses.push("Node.js");

// Add Element to the beginning
courses.unshift("Web Development");

console.log(courses);
```

6. Removing Elements from an Array

Remove elements using methods like pop(), shift(), or splice().

Ex :-

```
// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "Javascript", "React", "Node.js"];
console.log("Original Array: " + courses);

// Removes and returns the last element
let lastElement = courses.pop();
console.log("After Removing the last elements: " + courses);

// Removes and returns the first element
let firstElement = courses.shift();
console.log("After Removing the First elements: " + courses);
```

```
// Removes 2 elements starting from index 1
courses.splice(1, 2);
console.log("After Removing 2 elements starting from index 1: " + courses);
```

7. Array Length

Get the length of an array using the length property.

Ex :-

```
// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "Javascript", "React", "Node.js"];

let len = courses.length;

console.log("Array Length: " + len);
```

8. Increase and Decrease the Array Length

We can increase and decrease the array length using the JavaScript length property.

Ex :-

```
// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "Javascript", "React", "Node.js"];

// Increase the array length to 7
courses.length = 7;

console.log("Array After Increase the Length: ", courses);

// Decrease the array length to 2
courses.length = 2;
console.log("Array After Decrease the Length: ", courses)
```

9. Array Concatenation

Combine two or more arrays using the concat() method. It returns new array containing joined array elements.

Ex :-

```
// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "JavaScript", "React"];
let otherCourses = ["Node.js", "Expess.js"];

// Concatenate both arrays
let concateArray = courses.concat(otherCourses);

console.log("Concatenated Array: ", concateArray);
```

10. Conversion of an Array to String

We have a builtin method toString() to convert an array to a string.

Ex:-

```
// Creating an Array and Initializing with Values
let courses = ["HTML", "CSS", "JavaScript", "React"];

// Convert array to String
console.log(courses.toString());
```

11. Check the Type of an Arrays

The JavaScript typeof operator is used to check the type of an array. It returns “object” for arrays.

Ex :- concat , splice (operations) , increase and decrease of length

```
const arrone=["java","css","bootstrap",34,56,true];

console.log(arrone);

arrone.shift();

console.log(arrone);

arrone.pop();

console.log(arrone);


arrone.splice(2,1,78);

console.log(arrone.length);

arrone.length=3;

console.log(arrone.length);

console.log(arrone);

arrone.length=9;

console.log(arrone);

arrone.push("react");

arrone.push("nextjs");

arrone.push(67);

arrone.push(90);

console.log(arrone);

const arrtwo=[34,56,77,88,99];

const concatarray=arrtwo.concat(arrone);

console.log(concatarray);


const concatarrtwo=arrone.concat(arrtwo);
```

```
console.log(typeof(concatarrtwo));

console.log(concatarrtwo[4]);
```

When to use JavaScript Arrays and Objects?

- Arrays are used when we want element names to be numeric.
- Objects are used when we want element names to be strings.

Recognizing a JavaScript Array

There are two methods by which we can recognize a JavaScript array:

- **By using Array.isArray() method**
- **By using instanceof method**

Associative Arrays

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does not support arrays with named indexes.

In JavaScript, arrays always use numbered indexes.

Ex :-

```
const person = [];

person[0] = "John";
```

```
person[1] = "Doe";

person[2] = 46;

person.length;     // Will return 3

person[0];         // Will return "John"
```

Note :- If you use named indexes, JavaScript will redefine the array to an object.After that, some array methods and properties will produce incorrect results.

Objects :-

Real Life Objects :- In real life objects are like cars , house , animals , subject etc .

In JavaScript, an object is an unordered collection of key-value pairs. Each key-value pair is called a property.

JavaScript objects can be compared to real-world objects. They have properties and methods attached to them and properties are in the form of key-value pairs collections. Let us understand this more briefly with an example. In the real world, a car is an object and it has properties like name, color, price, model, etc. The car object also has some methods like start, breaks, stops, etc. All cars will have similar properties but the values will be different for each car. This same theory is applied in programming and is known as Object-Oriented Programming.

In JavaScript, objects are parent class which means they are king. If we understand objects, we understand JavaScript. As we know from the Data Types chapter. The primitive types of JavaScript are true, false, numbers, strings, null and undefined. Every other value is an object.

Everything is an object in JavaScript

1. There is no distinction between objects and classes
2. Objects inherit from objects
3. Even functions are objects

JavaScript is designed on a simple object-based paradigm(language). An object is a collection of properties. An object property is an association between a name (or key) and a value. A value of a property can be either.

1. Property (variable / field)

2. Function (method)

JavaScript variables are containers for data values. We can also treat objects like variables, but the only difference is that objects can contain many values whereas a variable has a single value at any given point of time. An object is a complex data type in JavaScript that is used to store any combination of data in a simple key-value pair. Each key-value pair is called a property. Data inside objects are unordered, and the values can be of any type.

JavaScript is template based not class-based. That means, in JavaScript, we don't have to create a class to get an object, instead we directly create objects in JavaScript.

The key of a property can be a string. The value of a property can be any value, e.g., a string, a number, an array, and even a function.

When an object has multiple properties, you use a comma (,) to separate them like the above example.

Ex :- Real life example

Car Object Properties Methods

car.name= suzuki	car.start()
car.model=500	car.drive()
car.weight=800kg	car.brake()
car.color=white	car.stop()

Object Properties :-

A real life car has properties like weight and color:

car.name = Fiat, car.model = 500, car.weight = 850kg, car.color = white.

Car objects have the same properties, but the values differ from car to car.

Object Methods :-

A real life car has methods like start and stop:

car.start(), car.drive(), car.brake(), car.stop().

Car objects have the same methods, but the methods are performed at different times.

Ex :- creating object with object literal and object constructor

```
// methods of defining an object
```

```
//1. with object literal

const person = {

    // add properties

    firstName:"Rashmi",

    lastName:"Hassin",

    age:30,

    78:"fair"

}

console.log(person[78]);



// 2. with new keyword

 const Employee = new Object({

    name:"John Doe",

    age:45,

    role:"SDE"

 });

 console.log(Employee.age);
```

JavaScript provides you with many ways to create an object. The most commonly used one is to use the object literal notation.

1. Ex :- const obj = {};

```
let person = {
  firstName: 'John',
  lastName: 'Doe'
};
```

The `person` object has two properties `firstName` and `lastName` with the corresponding values `'John'` and `'Doe'`.

2. Using new keyword or object constructor :-

Ex :- const person = new Object({

Name:”azmath”,

Age: 23,

Role:”developer

});

Accessing properties

To access a property of an object, you use one of two notations: the dot notation and array-like notation.

1) The dot notation (.)

The following illustrates how to use the dot notation to access a property of an object:

Syntax:- `objectName.propertyName`

For example, to access the `firstName` property of the `person` object, you use the following expression:

```
person.firstName
```

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};

console.log(person.firstName);
console.log(person.lastName);
```

2) Array-like notation ([])

The following illustrates how to access the value of an object's property via the array-like notation:

```
objectName['propertyName']
```

Ex :-

```
const Employee = new Object({
    name:"John Doe",
    age:45,
    role:"SDE"
 });
 console.log(Employee.age);


 console.log(Employee['name']); // accessing  obj property via rray
notation
```

When a property name contains spaces, you need to place it inside quotes. To access the `'building no'` property, you need to use the array-like notation , If you use the dot notation, you'll get an error:

Ex:-

```
const Employee = new Object({
    name:"John Doe",
    age:45,
    role:"SDE",
    'building no': "10-32/4D/D5"
 });
 console.log(Employee.age);
```

```
 console.log(Employee['name']); // accessing  obj property via array
notation

 console.log(Employee['building no']); // need to access via  array
notation we have space in property

//  console.log(Employee.building no); // error
```

Reading from a property that does not exist will result in an `undefined`.

For example:

```
const Employee = new Object({

    name:"John Doe",

    age:45,

    role:"SDE",

    'building no': "10-32/4D/D5"

 });

 console.log(Employee.age);
```


```
 console.log(Employee['name']); // accessing  obj property via array
notation

 console.log(Employee['building no']); // need to access via  array
notation we have space in property

//  console.log(Employee.building no); // error
```


```
console.log(Employee.district); // accessing property that is not defined
will give you undefined value
```

Modifying the value of a property

To change the value of a property, you use the assignment operator (=).

For example:

Ex :- use the same above example

```
Employee.name="azmath";

console.log(Employee);
```

Adding a new property to an object

Unlike objects in other programming languages such as Java and C#, you can add a property to an object after object creation.

```
Employee.city="Hyderabad";

console.log(Employee);
```

Deleting a property of an object

To delete a property of an object, you use the `delete` operator:

```
delete objectName.propertyName;
```

Ex :-

```
delete Employee.city;
```

```
console.log(Employee);
```

If you attempt to reaccess the city property, you'll get an `undefined` value.

Checking if a property exists :-

To check if a property exists in an object, you use the `in` operator:

`propertyName in objectName`

The `in` operator returns `true` if the `propertyName` exists in the `objectName`.

Ex :-

```
console.log('state' in Employee); // false it doest not exists
```

Note :-

- **An object is a collection of key-value pairs.**
- **Use the dot notation (`.`) or array-like notation (`[ ]`) to access the property of an object.**
- **Use the `delete` operator to remove a property from an object.**
- **Use the `in` operator to check if a property exists in an object.**

Ex :- (example discussed in class) 13 aug , 2024

```
const Employee= new Object({

    name:"Azmath",
```

```
    age:24,
    role:"developer"
});
// accessing property of an object via dot notation
console.log(Employee.name);
// accessing property of an object via array notation

console.log(Employee['age']);

const Address= new Object({
    street:"XYZ street",
    colony:"Rg road ",
    'building no': "Ag apartments"

});
console.log(Address['building no']); // accessing via array notation bucz it contains space between the 'building no'
console.log(Employee.state); // accesing the property that is not defined for an object will give u undefined value
console.log(Employee);
Employee.name="Vinay"; // replace the vale of a property
console.log(Employee);
 // accesing the property that is not defined for an object will give u undefined value
```

```
console.log(Employee.state);

 // adding a new property to an existing object


Employee.state="Telangana";

console.log(Employee.state);

// deleting a property of an object

delete Employee.state;

console.log(Employee.state);

// check whether the property exists  for an object or not -> it returns
boolean values

console.log('state' in Employee); // false - > does not exists

console.log('name' in Employee); // true - >  exists
```

Object Methods :- We provide methods for objects to describe its behavior and functions .

Ex :- 1

```
const car = new Object({

    name:"Suzuki",

    color:"black",

    speed:50
```

```
});

car.start= function(){

    console.log("first method of car");

    console.log(car);

}

car.start();
```

In this example:

- First, use a function expression to define a function and assign it to the `start` property of the `person` object.
- Then, call the method `start()` method.

Besides using a function expression, you can define a function and assign it to an object like this:

Ex:-

```
const car = new Object({

    name:"Suzuki",

    color:"black",

    speed:50

});


function stop(){

    console.log("second method of car");

}
```

```
car.stop=stop;

console.log(car);
```

- First, define the `stop()` function as a regular function.
- Second, assign the function name to the `stop` property of the `car` object.
- Third, call the `stop()` method.

Object method shorthand

JavaScript allows you to define methods of an object using the object literal syntax as shown in the following example:

Ex:-

```
const car = new Object({

    name:"Suzuki",

    color:"black",

    speed:50,

    brake: function(){

        console.log("third methd of car ");

    }

});


function stop(){

    console.log("second method of car");
```

```
}

car.stop=stop;

console.log(car);
```

ES6 provides you with the concise method syntax that allows you to define a method for an object:

Ex :-

```
const car = new Object({

    name:"Suzuki",

    color:"black",

    speed:50,

    brake: function(){

        console.log("third methd of car ");

    },

    drive (){

        console.log("fourth method of car");

    }

});


function stop(){

    console.log("second method of car");

}

car.stop=stop;

console.log(car);
```

```
car.brake();

car.drive();
```

This above code looks much cleaner and less verbose.

Ex:- function expression

```
// function expression

const start= function fun1(){

    console.log("function expression");

}

console.log(start);
```

this value :- In Javascript this keyword refers to an object .This keyword refers to different objects depending on how it is used .

- In object method , this refers to the object
- Alone , this refers to global object
- In function , this refers to the **global object**
- In a function , in strict mode , this is undefined
- In an event , this refers to the element that received the event
- Methods like call , apply , bind can refer this to any object .

Typically, methods need to access other properties of the object.

For example, you may want to define a method that returns the full name of the person object by concatenating the first name and last name.

Inside a method, the `this` value references the object that invokes the method. Therefore, you can access a property using the `this` value as follows

Syntax :- this.propertyName

Ex:-

```
const person = {
    firstName: 'John',
    lastName: 'Doe',
    greet: function () {
        console.log('Hello, World!');
    },
    getFullName: function () {
        return this.firstName + ' ' + this.lastName;
    }
};
console.log(person);
```

Note :- When a function is a property of an object, it becomes a method.

Ex :-

```
const person = {
    firstName: 'John',
    lastName: 'Doe',
    greet: function () {
        console.log('Hello, World!');
    },
    getFullName: function () {
        return this.firstName + ' ' + this.lastName;
    }
};
```

Nested Object (Nested Properties of an Object) :-

Nested properties of an object are properties within properties. An object can contain another object as a property. we can allocate another object as a property of an object. In other words, an object can have a property that is another object.

Values of an object's properties can be another object. We can access nested properties of an object or nested objects using the dot notation (.) or the bracket notation []. Nested properties of an object can be accessed by chaining key or properties names in the correct sequence order.

Ex :-

```
const person = {

    fname:"John",

    lname:"Doe",

    age: 24,

    address :{
```

```
        HNo :"10-32/5d",
        street: "XYZ street",
        colony:"ABC Colony",
        country:{
            state:"Telangana",
            country:"India"
        }
    }
}
console.log(person.address.colony);
// console.log(address.street);

console.log(person['address']['country']['state']);
```

Ex :- 2

```
const person = {
    fname:"John",
    lname:"Doe",
    age: 24,
    address :{
```

```
        HNo :"10-32/5d",
        street: "XYZ street",
        colony:"ABC Colony",
        country:{
            state:"Telangana",
            country:"India"
        }
    }
}
 var data= "address";
 var data2= "country";
 console.log(person[data]); // array notation
 console.log(person.data); // dot notation  --> undefined
```

Types of objects :-

There are two types of objects

- User - defined Objects
- Built - in Objects / Predefined Objects

User - defined Objects :- An object that is defined by the developer to develop a software object model , developers can create their own object such as Bank Account , Students , Product etc

Built - in Objects / Predefined Objects :- A built - in object is an object that is of the primitive and non - primitive data types , handling some specific tasks . These built-in objects are available and pre-defined by Javascript. Lots of predefined objects are available in Js . Commonly used Reference objects are as follows

Common Reference Objects :-

- Array Object
- Boolean Object
- Math Object
- Date Object
- Number Object
- String Object
- Regular Expression

DOM (Data Object Model)

- Attributes Object
- Element Object
- Style Object
- Document Object

BOM (Browser Object Model)

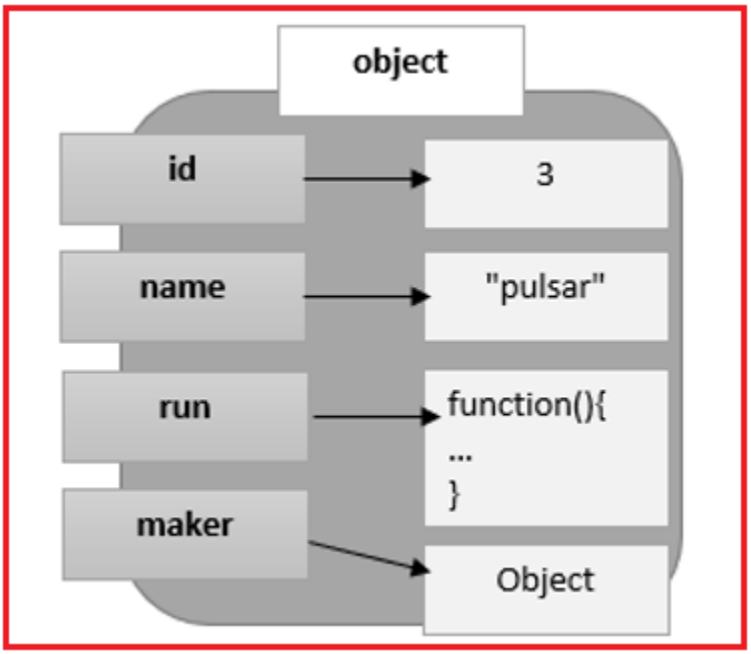
- Windows Object
- Navigator Object
- Location Object
- Screen Object
- Storage Object

Features of Javascript Object :-

The Object is an instance of class that has the structure of its blueprint but also owns its unique stat and behavior. Just like many other languages , Javascript is based on objects . Js Objects are simple maps from names (or keys) to

- values (properties)
- functions (methods)

Objects are containers with properties , properties can contain anything either functions or other objects .



Operators :- JavaScript Operators are symbols used to perform specific mathematical, comparison, assignment, and logical computations on operands. They are fundamental elements in JavaScript programming, allowing developers to manipulate data and control program flow efficiently. Understanding the different types of operators and how they work is important for writing effective and optimized JavaScript code. The various operators supported by Javascript are as follows

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators
- Ternary operators
- Relational operators
- Unary operators
- Comma Operators
- String operators

Task :- write one example for each of the operator types , read about operators, understand the operators and then complete examples for operators . Include all operators in examples .

Arithmetic (+ , – , % , / ,*)

Assignment (same as arithmetic with =)

Comparison (< , > ,<= , >= , != , == , ===)

Logical (AND ., OR , NOT)

Bitwise (AND , OR , XOR , NOT , Left shift , Right shift)

Ternary (:?)

Unary (typeof , delete)

Comma (,)

Relational (in , instanceof)

Operators :-

JavaScript operators are special symbols that perform operations on one or more operands (values). For example,

2 + 3 = 5 // here we used + operator to add 2 , 3 (operands)

1. **Arithmetic Operators :-** We use arithmetic operators to perform arithmetic calculations like addition, subtraction, etc. For example.

   Ex :- 5 - 3 = 2

Commonly Used Arithmetic Operators

Operator	Name	Example
+	Addition	`3 + 4  // 7`
-	Subtraction	`5 - 3  // 2`

*	Multiplication	2 * 3 // 6
/	Division	4 / 2 // 2
%	Remainder	5 % 2 // 1
++	Increment (increments by 1)	++5 or 5++ // 6
--	Decrement (decrements by 1)	--4 or 4-- // 3
**	Exponentiation (Power)	4 ** 2 // 16

Ex :- Arithmetic Operator

```
let b =3-7;
console.log(b);

let c =3*7;
console.log(c);

let d =21/7;
console.log(d);

let e =21%7;
console.log(e);


let a =3+7;
```

```
console.log(a);

let f=12;
// f++;
f--;
console.log(f);
let x= 5**3
console.log(x);
```

2. **Assignment Operators ;-** We use assignment operators to assign values to variables. For example,

   let x= 5;

   Here, we used the `=` operator to assign the value 5 to the variable `x`.

Commonly Used Assignment Operators

Operator	Name	Example
`=`	Assignment Operator	`a = 7;`
`+=`	Addition Assignment	`a += 5;  // a = a + 5`
`-=`	Subtraction Assignment	`a -= 2;  // a = a - 2`
`*=`	Multiplication Assignment	`a *= 3;  // a = a * 3`
`/=`	Division Assignment	`a /= 2;  // a = a / 2`

%=	Remainder Assignment	a %= 2; // a = a % 2
**=	Exponentiation Assignment	a **= 2; // a = a**2

Ex :- Assignment Operator

```
// assignment operator
let a = 7;
console.log("Assignment: a = 7, a =", a);

// addition assignment operator
a += 5;  // a = a + 5
console.log("Addition Assignment: a += 5, a =", a);

// subtraction assignment operator
a -= 5;  // a = a - 5
console.log("Subtraction Assignment: a -= 5, a =", a);

// multiplication assignment operator
a *= 2;  // a = a * 2
console.log("Multiplication Assignment: a *= 2, a =", a);

// division assignment operator
a /= 2;  // a = a / 2
console.log("Division Assignment: a /= 2, a =", a);

// remainder assignment operator
a %= 2;  // a = a % 2
console.log("Remainder Assignment: a %= 2, a =", a);

// exponentiation assignment operator
a **= 2;  // a = a**2
console.log("Exponentiation Assignment: a **= 7, a =", a);
```

3. **Comparison Operators :-** We use comparison operators to compare two values and return a boolean value (true or false). For example,

   Ex :-

```
const x = 3, b = 2;
console.log(x > b);

// Output: true
```

Here, we have used the `>` comparison operator to check whether `a` (whose value is 3) is greater than `b` (whose value is 2).

Since 3 is greater than 2, we get `true` as output.

Commonly Used Comparison Operators

Operator	Meaning	Example
`==`	Equal to	`3 == 5` gives us `false`
`!=`	Not equal to	`3 != 4` gives us `true`
`>`	Greater than	`4 > 4` gives us `false`
`<`	Less than	`3 < 3` gives us `false`

>=	Greater than or equal to	`4 >= 4` gives us `true`
<=	Less than or equal to	`3 <= 3` gives us `true`
===	Strictly equal to	`3 === "3"` gives us `false`
!==	Strictly not equal to	`3 !== "3"` gives us `true`

Ex :-

```
// equal to operator
console.log("Equal to: 2 == 2 is", 2 == 2);

// not equal operator
console.log("Not equal to: 3 != 3 is", 3 != 3);

// strictly equal to operator
console.log("Strictly equal to: 2 === '2' is", 2 === '2');

// strictly not equal to operator
console.log("Strictly not equal to: 2 !== '2' is", 2 !== '2');

// greater than operator
console.log("Greater than: 3 > 3 is", 3 > 3);

// less than operator
console.log("Less than: 2 > 2 is", 2 > 2);

// greater than or equal to operator
console.log("Greater than or equal to: 3 >= 3 is", 3 >= 3);

// less than or equal to operator
console.log("Less than or equal to: 2 <= 2 is", 2 <= 2);
```

4. **Logical Operators :-** We use logical operators to perform logical operations on boolean expressions. For example,

   Ex :-

```
const x = 5, y = 3;
console.log((x < 6) && (y < 5));
```

Here, `&&` is the logical operator AND. Since both `x < 6` and `y < 5` are `true`, the combined result is `true`.

Commonly Used Logical Operators

Operator	Syntax	Description
`&&` (Logical AND)	`expression1 && expression2`	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	`expression1 || expression2`	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	`!expression`	`false` if `expression` is `true` and vice versa

Ex :-

```
let x = 3;

// logical AND
console.log((x < 5) && (x > 0));  // true
console.log((x < 5) && (x > 6));  // false

// logical OR
```

```
console.log((x > 2) || (x > 5));  // true
console.log((x > 3) || (x < 0));  // false

// logical NOT
console.log(!(x == 3));  // false
console.log(!(x < 2));  // true
```

5. Bitwise Operators :-

We use bitwise operators to perform binary operations on integers.

Operator	Description	Example
&	Bitwise AND	`5 & 3  // 1`
|	Bitwise OR	`5 | 3  // 7`
^	Bitwise XOR	`5 ^ 3  // 6`
~	Bitwise NOT	`~5  // -6`
<<	Left shift	`5 << 1  // 10`
>>	Sign-propagating right shift	`-10 >> 1  // -5`

>>>	Zero-fill right shift	-10 >>> 1 // 2147483643

6. **String concatenation operator :-** In JavaScript, you can also use the + operator to concatenate (join) two strings. For example,

   Ex :-

```
let str1 = "Hel", str2 = "lo";
console.log(str1 + str2);

// Output: Hello
```

7. **JavaScript Miscellaneous Operators :-** JavaScript has many more operators besides the ones we listed above. You will learn about them in detail in later tutorials.

Operator	Description	Example
,	Comma: Evaluates multiple operands and returns the value of the last operand.	let a = (1, 3, 4); // 4
?:	Ternary: Returns value based on the condition.	(50 > 40) ? "pass" : "fail"; // "pass"
typeof	Returns the data type of the variable.	typeof 3; // "number"

`instanceof`	Returns `true` if the specified object is a valid object of the specified class.	`objectX instanceof ClassX`
`void`	Discards any expression's return value.	`void(x) // undefined`

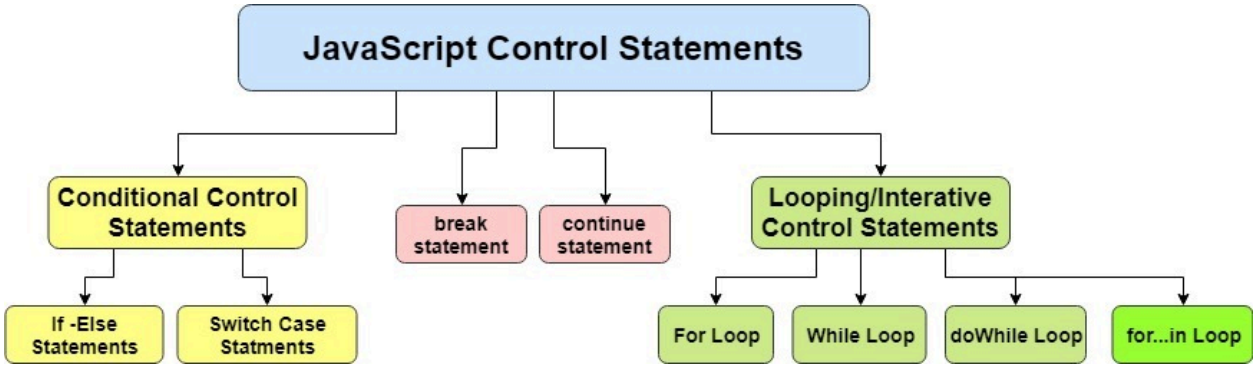
Control Statements :- JavaScript control statement is used to control the execution of a program based on a specific condition. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.

Types of Control Statements in JavaScript:-

Conditional Statement: These statements are used for decision-making, a decision , n is made by the conditional statement based on an expression that is passed. Either YES or NO.

Iterative Statement: This is a statement that iterates repeatedly until a condition is met. Simply said, if we have an expression, the statement will keep repeating itself until and unless it is satisfied.

There are several methods that can be used to perform control statements in JavaScript:

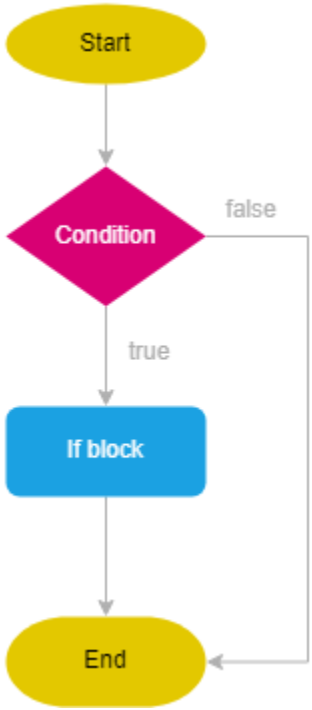


1. **if :-** The `if` statement executes block if a condition is `true`. The following shows the syntax of the `if` statement:

```
if( condition )

    statement;
```

The condition can be a value or an expression. Typically, the condition evaluates to a Boolean value, which is true or false.

If the condition evaluates to true, the if statement executes the statement. Otherwise, the if statement passes the control to the next statement after it. The following flowchart illustrates how the if statement works:



Ex:-

```
let age = 18;
if (age >= 18) {
  console.log('You can sign up');
}
```

First, declare and initialize the variable age to 18

Second, check if the age is greater or equal to 18 using the if statement. Because the expression age >= 18 is true, the code inside the if statement executes that outputs a message to the console

The following example also uses the if statement. However, the age is 16 which causes the condition to be evaluated to false. Therefore, you won't see any message in the output

2. **Nested if :-** It's possible to use an `if` statement inside another `if` statement. For example

```
let name = 'Rahul';

let password = 'Rahul@1236';


if (name == 'Rahul') {

  if (password.length <= 9) {

    console.log('Login Success');

  }

}
```

In practice, you should avoid using nested `if` statements as much as possible. For example, you can use the && operator to combine the conditions and use an `if` statements as follows

```
let names = 'Rahul';

let password = 'Rahul@1236';
```

```
if (names == 'Rahul' && password.length <= 10) {


    console.log('Login Success');


}
```

Note :-

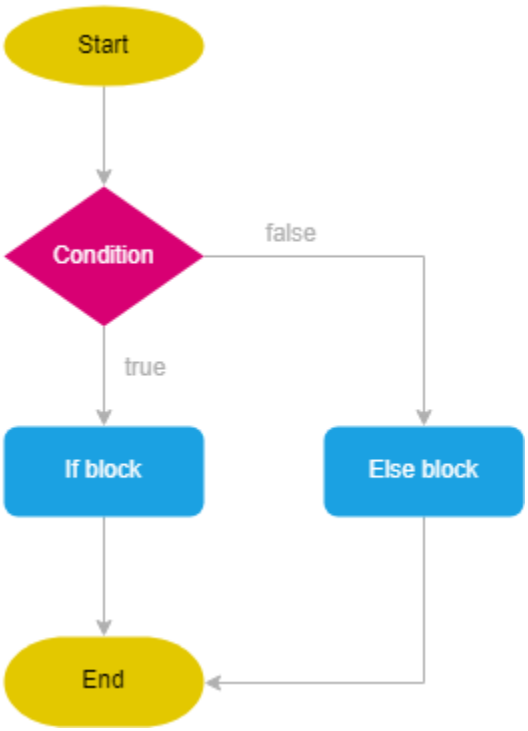
- **Use the JavaScript `if` statement to execute a statement if a condition is `true`.**
- **Avoid using nested `if` statements as much as possible.**

3. **If - else :-** The `if` statement executes a block if a condition is `true`. When the condition is `false`, it does nothing. But if you want to execute a statement if the condition is `false`, you can use an `if...else` statement.

   The following shows the syntax of the `if...else` statement:

```
if( condition ) {
    // ...
  } else {
    // ...
  }
```

The following flowchart illustrates how the `if...else` statement works:



Ex :-

```
let names = 'Rahul';

let password = 'Rahul@1236';


if (names == 'Rahul' && password.length <= 9) {


    console.log("Login Success");


}


else{
```

```
    console.log("Incorrect Credentials")
}
```

Note :-

- **Use the JavaScript `if...else` statement to execute a block if a condition is `true` and another block otherwise.**

3. if-else if :- The if an if...else statement accepts a single condition and executes the if or else block accordingly based on the condition.

To check multiple conditions and execute the corresponding block if a condition is true, you use the if...else...if statement like this:

```
if (condition1) {
    // ...
  } else if (condition2) {
    // ...
  } else if (condition3) {
    //...
  } else {
    //...
  }
```

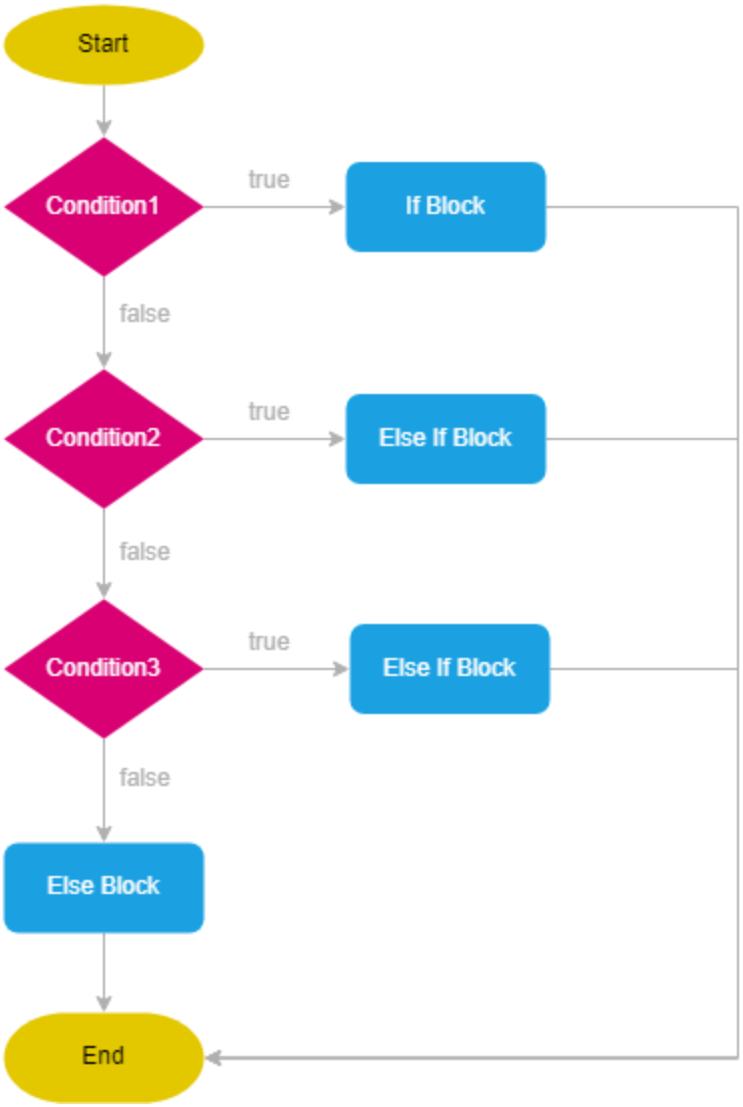
In this syntax, the `if...else...if` statement has three conditions. In theory, you can have as many conditions as you want to, where each `else...if` branch has a condition.

The `if...else...if` statement checks the conditions from top to bottom and executes the corresponding block if the condition is `true`.

The `if...else...if` statement stops evaluating the remaining conditions as soon as a condition is `true`. For example, if the `condition2` is `true`, the `if...else...if` statement won't evaluate the `condition3`.

If all the conditions are `false`, the `if...else...if` statement executes the block in the `else` branch.

The following flowchart illustrates how the `if...else...if` statement works:



Ex :-

```
let month = 6;

let monthName;


if (month == 1) {

  monthName = 'Jan';

} else if (month == 2) {
```

```
    monthName = 'Feb';
  } else if (month == 3) {
    monthName = 'Mar';
  } else if (month == 4) {
    monthName = 'Apr';
  } else if (month == 5) {
    monthName = 'May';
  } else if (month == 6) {
    monthName = 'Jun';
  } else if (month == 7) {
    monthName = 'Jul';
  } else if (month == 8) {
    monthName = 'Aug';
  } else if (month == 9) {
    monthName = 'Sep';
  } else if (month == 10) {
    monthName = 'Oct';
  } else if (month == 11) {
    monthName = 'Nov';
  } else if (month == 12) {
    monthName = 'Dec';
  } else {
    monthName = 'Invalid month';
```

```
}

console.log(monthName);
```

In this example, we compare the month with 12 numbers from 1 to 12 and assign the corresponding month name to the monthName variable.

Since the month is 6, the expression month==6 evaluates to true. Therefore, the if...else...if statement assigns the literal string 'Jun' to the monthName variable. Therefore, you see Jun in the console.

If you change the month to a number that is not between 1 and 12, you'll see the Invalid Month in the console because the else clause will execute.

Note :-

- **Use the JavaScript `if...else...if` statement to check multiple conditions and execute the corresponding block if a condition is `true`.**

Disadvantages of nested if else if :-

- Deeply nested code , hard to understand
- Increases Complexity , more prone to errors and bugs
- Slower Execution , more conditions to evaluate
- Limited Scalability , need to add further nesting for adding new conditions .

To overcome all these issues we go for Switch Statement

4. **Switch :-** The switch statement evaluates an expression, compares its results with case values, and executes the statement associated with the matching case value.

   The following illustrates the syntax of the switch statement:

```
switch (expression) {
  case value1:
      statement1;
      break;
  case value2:
      statement2;
      break;
  case value3:
      statement3;
      break;
  default:
      statement;
}
```

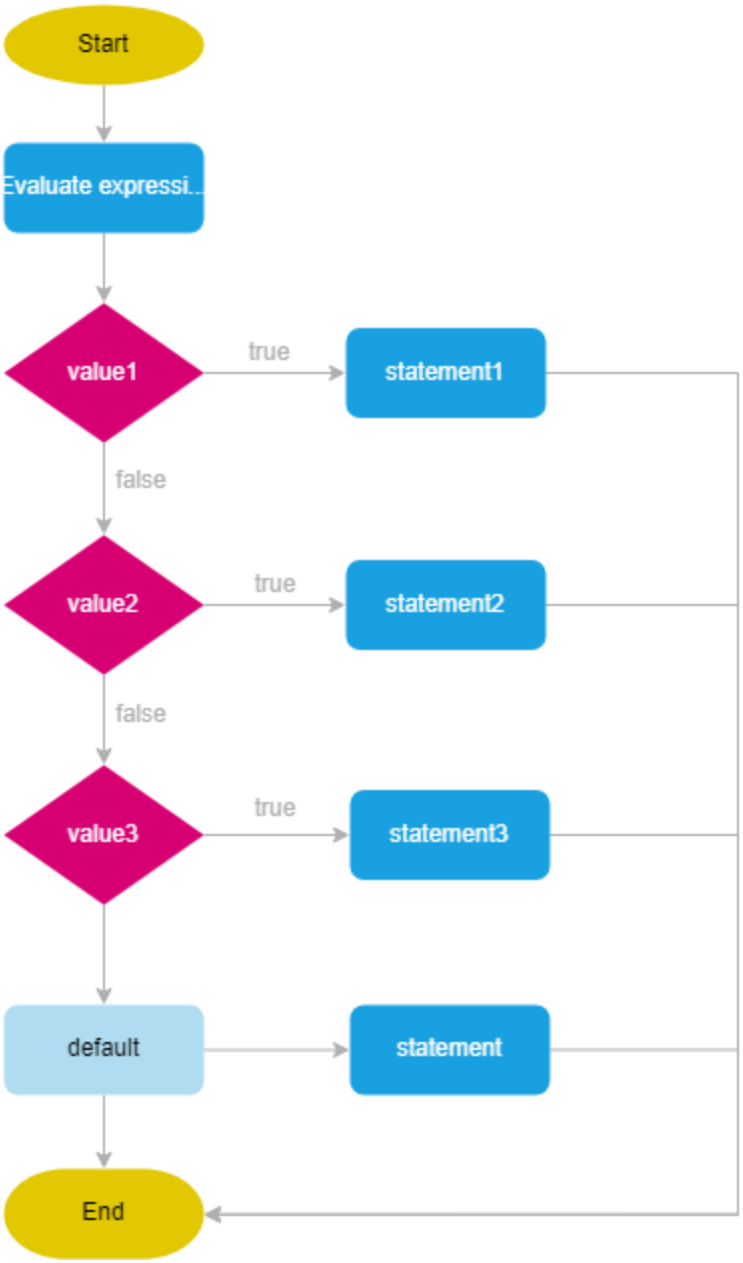
- First, evaluate the `expression` inside the parentheses after the `switch` keyword.
- Second, compare the result of the expression with the `value1`, `value2`, … in the `case` branches from top to bottom. The `switch` statement uses strict comparison (===).

- Third, execute the statement in the `case` branch where the result of the `expression` equals the value that follows the `case` keyword. The `break` statement exits the `switch` statement. If you omit the `break` statement, the code execution falls through the original case branch into the next one. If the result of the expression does not strictly equal any value, the `switch` statement will execute the `statement` in the `default` branch.

The `switch` statement will stop comparing the `expression`'s result with the remaining case values as long as it finds a match.

The `switch` statement is like the if…else…if statement. But it has more readable syntax.

The following flowchart illustrates the `switch` statement:



Ex :-

```
let day = 3;

let dayName;


switch (day) {
```

```
case 1:
  dayName = 'Sunday';
  break;
case 2:
  dayName = 'Monday';
  break;
case 3:
  dayName = 'Tuesday';
  break;
case 4:
  dayName = 'Wednesday';
  break;
case 5:
  dayName = 'Thursday';
  break;
case 6:
  dayName = 'Friday';
  break;
case 7:
  dayName = 'Saturday';
  break;
default:
  dayName = 'Invalid day';
```

```
}


console.log(dayName); // Tuesday
```

Note :-

- **The `switch` statement evaluates an expression, compares its result with `case` values, and executes the statement associated with the matching case.**
- **Use the `switch` statement rather than a complex `if...else...if` statement to make the code more readable.**
- **The `switch` statement uses the strict comparison (===) to compare the `expression` with the `case` values.**

Iterative Statements :-

1. **for loop :-** for loop is used to execute a block of code repeatedly for a specified number of times .

   The `for` loop statement creates a loop with three optional expressions. The following illustrates the syntax of the `for` loop statement:

   for (initializer; condition; iterator) {

   // statements

   }

initializer

The `for` statement executes the `initializer` only once the loop starts. Typically, you declare and initialize a local loop variable in the initializer.

2) **condition**

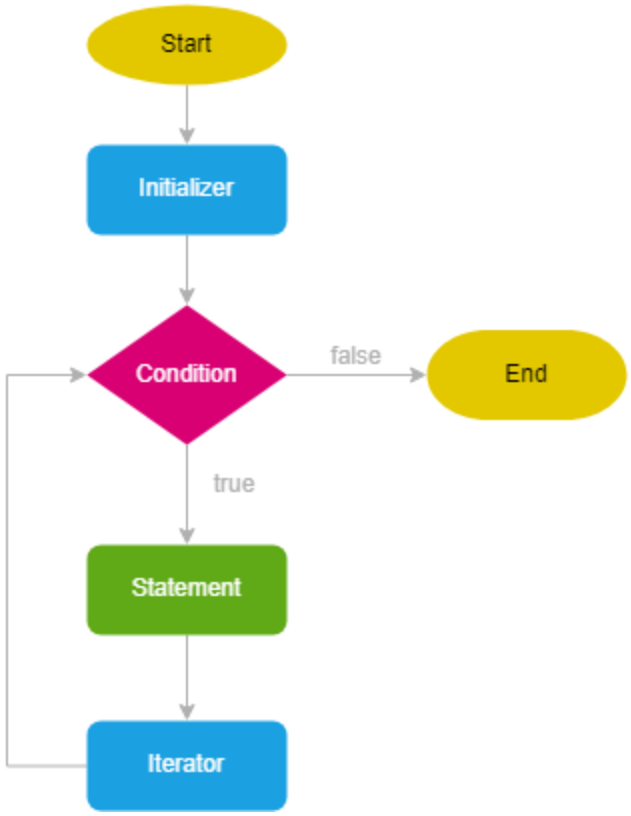
The `condition` is a boolean expression that determines whether the `for` should execute the next iteration.

The `for` statement evaluates the `condition` before each iteration. If the condition is `true` (or is not present), it executes the next iteration. Otherwise, it'll end the loop.

3) iterator

The `for` statement executes the `iterator` after each iteration.

The following flowchart illustrates the `for` loop:



In the `for` loop, the three expressions are optional. The following shows the `for` loop without any expressions:

```
for ( ; ; ) {
    // statements
 }
```

Ex:-

```
for (let i = 1; i < 5; i++) {
    console.log(i);
  }
```

How it works.

- First, declare a variable `counter` and initialize it to 1.

- Second, display the value of the counter in the console if the counter is less than 5.
- Third, increase the value of `counter` by one in each iteration of the loop

2. **Using for loop without initializer example :-**

```
let j = 1;

for (; j < 10; j += 2) {

  console.log(j);

}
```

3. **Using for loop without condition example :-**

```
for (let j = 1; ; j += 2) {

    console.log(j);

    if (j > 10) {

      break;

    }

  }
```

4. **Using for loop without any expression example :-**

```
let j = 1;

for (;;) {

  if (j > 10) {

    break;

  }

  console.log(j);
```

```
  j += 2;
}
```

5. **Using for loop without any loop body example :-**

```
let sum = 0;

for (let i = 0; i <= 9; i++, sum += i);

console.log(sum);
```

Note :- Use the JavaScript `for` statement to create a loop that executes a block of code using various options.

2. While loop :- The JavaScript `while` statement creates a loop that executes a block as long as a condition evaluates to `true`.

The following illustrates the syntax of the `while` statement:

```
while (expression) {
    // statement
}
```

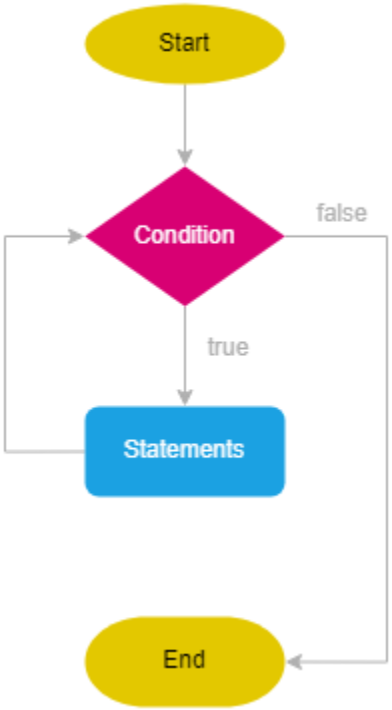
The `while` statement evaluates the `expression` before each iteration of the loop.

If the `expression` evaluates to `true`, the `while` statement executes the `statement`. Otherwise, the `while` loop exits.

Because the `while` loop evaluates the `expression` before each iteration, it is known as a pretest loop.

If the `expression` evaluates to `false` before the loop enters, the `while` loop will never execute.

The following flowchart illustrates the `while` loop statement:



> **Note that if you want to execute the statement at least once and check the condition after each iteration, you should use the do...while statement.**

Ex :- The following example uses the `while` statement to output the odd numbers between 1 and 10 to the console:

```
let count = 1;

while (count < 10) {

    console.log(count);
```

```
    count +=2;
}
```

How the script works

- First, declare and initialize the `count` variable to 1.
- Second, execute the statement inside the loop if the `count` variable is less than `10`. In each iteration, output the count to the console and increase the count by 2.
- Third, after 5 iterations, the `count` is 11. Therefore, if the condition `count < 10` is `false`, the loop exits.

Note :- Use a `while` loop statement to create a loop that executes a block as long as a `condition` evaluates to `true`.

Do while loop :- The `do...while` loop statement creates a loop that executes a block until a condition evaluates to `false`. The following statement illustrates the syntax of the `do...while` loop:

```
do {
    statement;
  } while(expression);
```

Unlike the `while` loop, the `do-while` loop always executes the `statement` at least once before evaluating the `expression`.

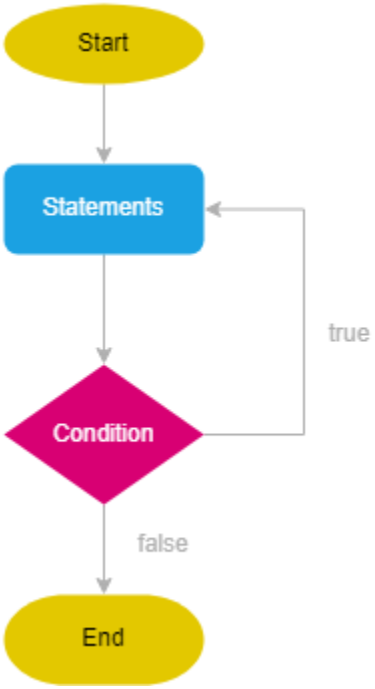
Because the `do...while` loop evaluates expression after each iteration, it's often referred to as a post-test loop.

Inside the loop body, you need to make changes to some variables to ensure that the `expression` is `false` after some iterations. Otherwise, you'll have an indefinite loop.

Note that starting with ES6+, the trailing semicolon (`;`) after the `while(expression)` is optional. So you can use the following syntax:

```
do {
    statement;
} while(expression)
```

The following flowchart illustrates the `do-while` loop statement:



In practice, you often use the `do...while` statement when you want to execute the loop body at least once before checking the condition.

The following example uses the `do...while` statement to output five numbers from 0 to 4 to the console:

Ex:- 1

```
let count = 0;

do {

  console.log(count);

  count++;

} while (count < 5)
```

In this example:

- First, declare and initialize the `count` variable to zero.
- Second, show the `count` and increase its value by one in each iteration until its value is greater or equal to 5.

Note:- Use the do…`while` statement to create a loop that executes a code block until a condition is `false`.

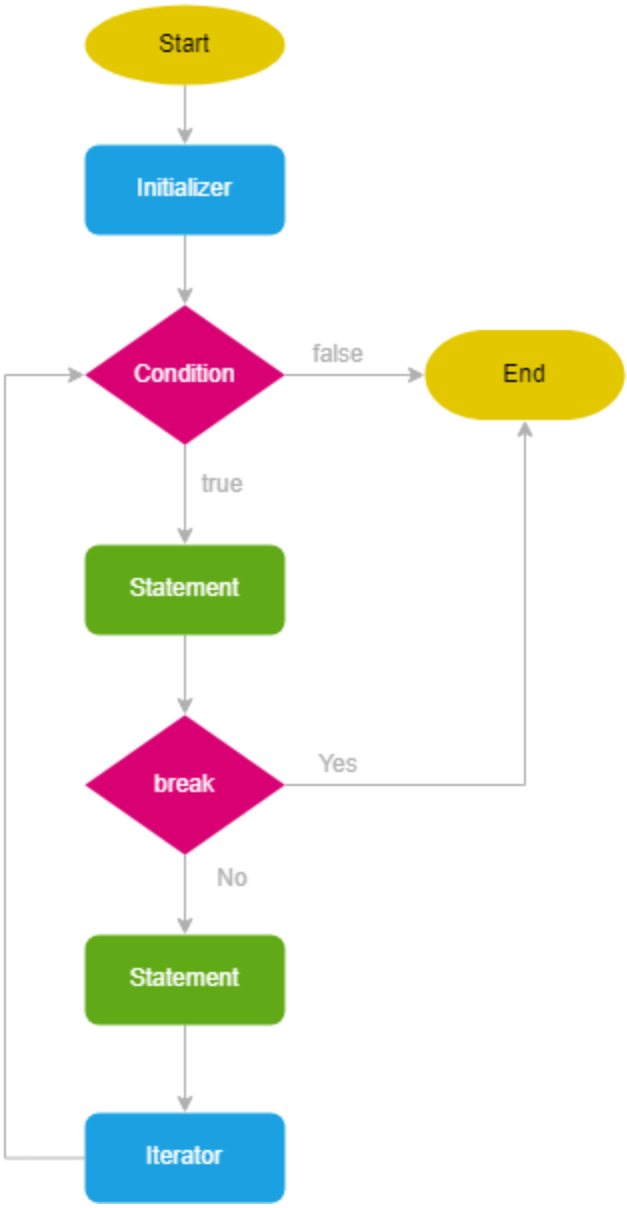
break:- The `break` statement prematurely terminates a loop such as `for`, `do...while`, and `while` loop, a `switch`, or a `label` statement. Here's the syntax of the `break` statement:

In this syntax, the `label` is optional if you use the `break` statement in a loop or `switch`. However, if you use the `break` statement with a label statement, you need to specify it.

```
for (let i = 0; i < 5; i++) {

    console.log(i);

    if (i == 2) {

      break;

    }

  }
```

In this example, we use an `if` statement inside the loop. If the current value of `i` is 2, the `if` statement executes the `break` statement that terminates the loop.

This flowchart illustrates how the `break` statement works in a `for` loop:



continue :- The `continue` statement terminates the execution of the statement in the current iteration of a loop such as a for, while, and do…while loop and immediately continues to the next iteration.

Here's the syntax of the `continue` statement:

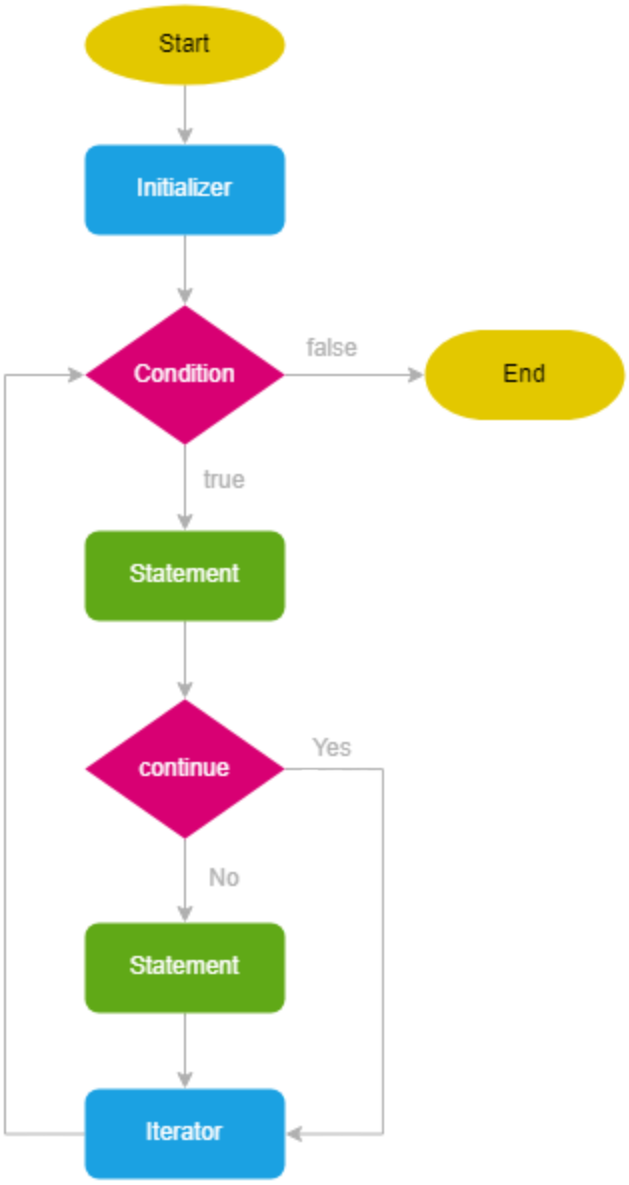
In this syntax, the label is optional. It is a valid identifier associated with the label of a statement. Read the `break` statement tutorial for more information on the label statement.

Typically, you use the `continue` with an `if` statement like this:

```
// inside a loop

if(condition){

    continue;

  }
```

When using the `continue` statement in a `for` loop, it doesn't terminate the loop entirely. Instead, it jumps to the `iterator` expression.

The following flowchart illustrates how the `continue` statement works in a `for` loop:



The following example uses a `continue` in a `for` loop to display the odd number in the console:

Ex :-

```
for (let i = 0; i < 10; i++) {

    if (i % 2 === 0) {

      continue;
```

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

The `i%2` returns the remainder of the division of the current value of `i` by 2.

If the remainder is zero, the `if` statement executes the `continue` statement that skips the current iteration of the loop and jumps to the iterator expression `i++`. Otherwise, it outputs the value of `i` to the console.

Ex:- 2

```
let i = 0;
while (i < 10) {
  i++;
  if (i % 2 === 0) {
    continue;
  }
  console.log(i);
}
```

Note :- Use the JavaScript `continue` statement to skip the current iteration of a loop and continue the next one.

Functions:- A function in JavaScript is a reusable block of code that performs a specific task. You define it once, and then you can run (or "call") it whenever you need that task done in your program.

Syntax;-

```
function functionName(Parameter1, Parameter2, ...)
{
    // Function body
}
```

To create a function in JavaScript, we have to first use the **keyword *function***, separated by the name of the function and parameters within parenthesis. The part of the function inside the curly braces **{}** is the body of the function.

Why Functions?

- Functions can be used multiple times, reducing redundancy.
- Break down complex problems into manageable pieces.
- Manage complexity by hiding implementation details.
- Can call themselves to solve problems recursively.

Function Invocation

The function code you have written will be executed whenever it is called.

- Triggered by an event (e.g., a button click by a user).
- When explicitly called from JavaScript code.

- Automatically executed, such as in self-invoking functions.

Function Definition

Before, using a user-defined function in JavaScript we have to create one. We can use the above syntax to create a function in JavaScript.

A function definition is sometimes also termed a function declaration or function statement. Below are the rules for creating a function in JavaScript:

- Every function should begin with the keyword *function* followed by,
- A user-defined function name that should be unique,
- A list of parameters enclosed within parentheses and separated by commas,
- A list of statements composing the body of the function enclosed within curly braces {}.

Ex :-

```
function add(number1, number2) {
    return number1 + number2;
}
console.log(add(6,9));
```

In the above example, we have created a function named **add.**

- This function accepts two numbers as parameters and returns the addition of these two numbers.
- Accessing the function with just the function name without () will return the function object instead of the function result.

Ex ;- 2

The following declares a function `say()` that accepts no parameter:

```
function say() {
}
```

Ex :- 3

The following declares a function named `square()` that accepts one parameter:

```
function square(number){
    return number*number;
}
console.log(square(10));
```

Calling a function:-

To use a function, you need to call it. Calling a function is also known as invoking a function. To call a function, you use its name followed by arguments enclosing in parentheses like this:

```
functionName(arguments);
```

Parameters vs. Arguments:-

The terms parameters and arguments are often used interchangeably. However, they are essentially different.

When declaring a function, you specify the parameters. However, when calling a function, you pass the arguments that are corresponding to the parameters.

For example, in the `say()` function, the `message` is the parameter and the `'Hello'` string is an argument that corresponds to the `message` parameter.

Returning a value:-

Every function in JavaScript implicitly returns `undefined` unless you explicitly specify a return value. For example:

```
function say(message="no corresponding value passed"){

    // console.log("hello ");

    return message;

}

console.log(say());
```

The arguments object

Inside a function, you can access an object called `arguments` that represents the named arguments of the function.

The `arguments` object behaves like an array though it is not an instance of the Array type.

For example, you can use the square bracket `[]` to access the arguments: `arguments[0]` returns the first argument, `arguments[1]` returns the second one, and so on.

Also, you can use the `length` property of the `arguments` object to determine the number of arguments.

Ex :-

```
function add() {
    let sum = 0;
    for (let i = 0; i < arguments.length; i++) {
        sum += arguments[i];
    }
    return sum;
}
console.log(add(1,2,3,7,8,9));
```

Hence, you can pass any number of arguments to the add( ) function.

Function hoisting:- In JavaScript, you can use a function before declaring it. For example

```
showMe(); // a hoisting example

function showMe(){
    console.log('an hoisting example');
}
```

This feature is called hoisting.

Function hoisting is a mechanism in which the JavaScript engine physically moves function declarations to the top of the code before executing them.

The following shows the version of the code before the JavaScript engine executes it:

```
function showMe(){
    console.log('a hoisting example');
}

showMe(); // a hoisting example
```

Note :-

- **Use the `function` keyword to declare a function.**
- **Use the `functionName()` to call a function.**
- **All functions implicitly return undefined if they don't explicitly return a value.**
- **Use the `return` statement to return a value from a function explicitly.**
- **The `arguments` variable is an array-like object inside a function, representing function arguments.**
- **The function hoisting allows you to call a function before declaring it.**

Anonymous Functions :- A function without a name is called as Anonymous functions . It is not accessible after its initial creation . it can only be accessed as a variable it is stored in as a function as a value. An anonymous function can also have multiple arguments, but only one expression.

Declaration:- There are two ways of declaring it

1)First method:

```
function () {
    function body
}
```

2)Second method:

We can also declare it by using arrow function technique

```
( ()=> {
// Function Body...
```

})();

example:- we can pass anonymous function as parameter to another function.

```
function f1(a){
a();
}
f1(function () {
console.log(“I am Anonymous function”);
})
```

example:- we can define anonymous function and store it in a variable , and access it by invoking the function like variable_name();

```
var greet= function (){
console.log(“I am anonymous function”);
};
greet();
```

Function Expression

Definition:-

The Javascript Function Expression is used to define a function inside any expression. The Function Expression allows

us to create an anonymous function that doesn’t have any function name which is the main difference between Function Expression and Function Declaration.

If we can refer a function with a variable, then it is known as a function expression.

example:- referring a function with a variable

```
var m=function add(a,b){
var c=a+b;
console.log(c);
console.log(“I am a Function Expression”;);
}
console.log(m(10,20));
```

example:- creating anonymous function

```
var sample= function(){
console.log(&amp;quot;I am function expression &amp;quot;);
}
sample();
```

Calling a Function:-

Just by declaring a function will not execute it we need to call it to use its functionality.

A function can be called by its name followed by parenthesis and semicolon

function_name();

// function having parameters

function_name(parameter 1, parameter2.....);

->Functions must be in scope when they are called.

->The arguments of a function are not limited to strings and numbers. You can pass whole objects to a function.

Function Scope:-

Variables defined inside a function cannot be accessed from anywhere outside the function, because the variable is defined

only in the scope of the function. However, a function can access all variables .

In other words, a function defined in the global scope can access all variables defined in the global scope. A function defined inside another function can also access all variables defined in its parent function, and any other variables to which the parent function has access.

Note:-

Important note:-Function hoisting only works with function declarations — not with function expressions.

First Class Function:-

Definition:-Javascript supports first class functions which means, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable.

1. can be referred with a variable

```
var auth = function(username, password){
// logic
}
```

example:-

```
var m=function add(a,b){
var c=a+b;
console.log(c);
console.log("I am a Function Expression");
}
console.log(m(10,20));
```

2. can be passed as an argument.

```
function f1(){
```

}

f1(function(){

})

example:-

function f1(a){

a();

}

f1(function () {

console.log(“I am Anonymous function”);

})

3. can also be returned from another function.

function f1(){

// logic

return function(){

console.log(“I am another function”);

}

}

Ex:-

function greetings(){

```
return function greet(){
console.log(“ Hello good morning”):
}
}
var m=greetings();
m();
```

IIFE:(Immediately Invoked Function Expression)

Definition:- An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined.

The name IIFE is promoted by Ben Alman.

Declaration:-

```
(function(){
console.log(“I am IIFe one”);
})();
```

We can have any number of IIFE’s in an application however each IIFE will be invoked only once.

```
(function () {
console.log(“I am IIFE1”);
})();
```

```
(function () {
console.log(“I am IIFE2”);
})();

(function () {
console.log(“I am IIFE3”);
})();
```

some more information about IIFE

It is a design pattern which is also known as a **Self-Executing Anonymous Function.**

It contains two major parts

1)The first is the anonymous function with lexical scope enclosed within the Grouping Operator (). This prevents accessing variables within the IIFE idiom as well as polluting the global scope.

2)The second part creates the immediately invoked function expression () through which the JavaScript engine will directly interpret the function.

Passing an Object as argument for a function :-

Ex:-

```
var person={
    name:"Rahul",
    age:30
}
function greet(obj){
    console.log(obj.name+"\t"+obj.age);
}
greet(person);
```

Call , Apply , Bind Functions :- (Related to objects concept)

Call :- If we want to reuse the existing object resources (when matching the structure), then use a call function.

Ex:-

```
var customer={
    name:"Rahul",
    address:"HYD",
    details: function(){
        return this.name+"\t"+this.address
    }
```

```
}
console.log(customer.details());
var customer2={
    name:"Kumar",
    address:"Delhi"
}
// access the details() -> method of Customer (obj) in Customer2 (obj)
//console.log(customer2.details());

console.log(customer.details.call(customer2));
```

Apply :- works same as call function but apply function helps the array auto splitting.

Ex:-

```
var emp ={
    name:"Rahul",
    address:"HYD",
    getInfo: function(country , pincode , hno) {
         return this.name +"\t" + this.address +"\t" + country +"\t" +
        pincode+"\t"+ hno;

    }
```

```
}
var emp2={
    name:"John",
    address:"Delhi",
}
console.log(emp.getInfo());
var arr= ["India" , 40449 , "12-09-pph"];
console.log(emp.getInfo.call(emp2,arr[0],arr[1],arr[2]));
```

Bind :- Bind:-The bind () helps the object to borrow a method from another object .

here we have two objects person and member

person object have fullname method and it is borrowed by

member object with help of bind ()

Ex :-

```
var emp ={
    name:"Rahul",
    address:"HYD",
    getInfo: function() {
        console.log(this.name +"\t" + this.address) ;
   }
```

```
}

var emp2={

    name:"John",

    address:"Delhi",



}

var hello=emp.getInfo.bind(emp2);

hello();
```

Call by value / Pass by value :-

In this method, values of actual parameters are copied to the function's formal parameters, and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in the actual parameters of the caller.

Suppose there is a variable named **"a"**. Now, we store a primitive value(boolean, integer, float, etc) in the variable **"a"**. Let us store an integer value in **"a"**, Let a=5. Now the variable **"a"** stores 5 and has an address location where that primitive value sits in memory.

Now, suppose we copy the value of **"a"** in **"b"** by assignment (**a=b**). Now, **"b"** points to a new location in memory, containing the same data as variable **"a"**.

Thus, a=b=5 but both point to separate locations in memory.

This approach is called **call by value** where 2 variables become the same by copying the value but in 2 separate spots in the memory.

Features of call by value:

- Function arguments are always passed by value.
- It copies the value of a variable passed in a function to a local variable.
- Both these variables occupy separate locations in memory. Thus, if changes are made in a particular variable it does not affect the other one.

Ex :-

```
let a=5; // 101xyz

let b;  // 102xyz

b=a; // copy value of a in to b var

a=3;

console.log(a); // 3

console.log(b); // 5
```

Ex :-

```
var y=100;

function f1(x){


    console.log(x); // 100

    x=200;
```

```
    console.log(x); // 200
}
f1(y);
console.log(y); // 100
```

“b” was just a copy of **“a”**. It has its own space in memory. When we change **“a”** it does not have any impact on the value of **“b”**.

Call by Reference :- Let’s say, we have an object stored in the variable “a”. The variable stores the location or the address where the object lives. Now we set b=a. Now that new variable “b” instead of pointing to a new location in the memory, points to the same location where “a” does. No new object is created, and no copy is created. Both variables point to the same object. This is like having 2 names.

This is called by reference. It behaves quite differently from value. All objects interact by reference.

Features of By reference:

- In JavaScript, all objects interact by reference.
- If an object is stored in a variable and that variable is made equal to another variable then both of them occupy the same location in memory.
- Changes in one object variable affect the other object variable.

Ex:-

```
let c = { greetings:"welcome" };

let d;

d=c;

c.greetings= "welcome to login";

console.log(c); //  welcome to login

console.log(d); //  welcome to login
```

String Interpolation (Es 6 topic) :-

String interpolation is a great programming language feature that allows injecting variables, function calls, and arithmetic expressions directly into a string. String interpolation was absent in JavaScript before ES6. String interpolation is a new feature of ES6, that can make multi-line strings without the need for an escape character. We can use apostrophes and quotes easily so that they can make our strings and therefore our code easier to read as well. These are some of the reasons to use string interpolation over string concatenation.

Ex :-

```
// String Concatenation

function myInfo(fname, lname, country) {

    return "My name is " + fname + " " + lname + ". "
```

```
            + country + " is my favorite country.";

}

console.log(myInfo("john", "doe", "India"));
```

Explanation :-

In string concatenation, it is hard to maintain strings as they grow large; it becomes tedious and complex. In order to make it readable, the developer has to maintain all the whitespaces. This is where ES6 comes to the rescue with String interpolation. In JavaScript, the template literals (strings wrapped in backticks ` `) and ${expression} as placeholders perform the string interpolation. Now we can write the above myInfo function with string interpolation.

Ex :- 2 (The below code shows String Interpolation in JavaScript)

```
// String Interpolation

function myInfo(fname, lname, country) {

    return `My name is ${fname} ${lname}. ${country} is my favorite
country`;

}

console.log(myInfo("john", "doe", "India"));
```

Explanation :- We can see that the code is small and easily readable as compared to concatenation. The template string supports placeholders. The expression like variables, function call, arithmetic can be placed inside the placeholder. These expressions are evaluated at runtime and output is inserted in the string.

Ex :- 3

```
const x = "w3wiki";



// I like w3wiki

console.log(`I like ${x}.`);



// Function call

function greet() {

    return "hello!";

}



// hello! I am a student.

console.log(`${greet()} I am a student.`);



// Expression evolution
```

```
//sum of 5 and 6 is 11.

console.log(`sum of 5 and 6 is ${5+6}.`);
```

Ex :- 4 (conditional statements in Expression)

```
function isEven(x) {

    console.log(`x is ${x%2 == 0 ? 'even': 'odd'}`);

}

isEven(4); // x is even
```

Note :- The string interpolation is a great feature. It helps in inserting values into string literals. It makes code more readable and avoids clumsiness. (Es 6)

Expression substitution (String replace method) :-

In javascript replace() method is a string method that returns a new string with some or all matches of a pattern replaced by a replacement string . The original string remains unchanged .

Syntax :- str.replace(original string , new string / replacement string)

Ex :-

```
let text = "Visit Microsoft!";

let result = text.replace("Microsoft", "Google");

console.log(result);
```

Ex :- 2

```
// replace multiple values in a string

let news = " Heavy Rainfall for four days !"

let latestnews = news.replace('Heavy','High Alert')

                    .replace('Rainfall','due to rains');

console.log(latestnews);
```

Operator Precedence :-

Operator precedence refers to the priority given to operators while parsing a statement that has more than one operator performing operations in it. Operators with higher priorities are resolved first. But as one goes down the list, the priority decreases and hence their resolution.

```
( * ) and ( / ) have higher precedence than ( + ) and ( - )
```

Precedence and Associativity

The associativity represents which operator has to solve first while going from left to right if two or more operators have the same precedence in the expression.

Ex :- `2 + 3 * 4 / 3 =  2 + (3 * 4) / 3  // 6`

The precedence of the multiply(*) and divide(/) operators are the same but due to associativity from left to right the multiply operator will be resolved first as it comes first when we go from left to right and hence the result will be 6.

Operator Precedence and Associativity Table :-

The operator precedence and associativity table can help one know the precedence of an operator relative to other operators. As one goes down the table, the precedence of these operators decreases over each other. The operators as subparts of precedence will have the same specified precedence and associativity as contained by the main part.

Precedence	Operator	Description	Associativity	Example
1	()	Grouping	–	(1+2)
2	.	Member	left to right	obj.function

2.1	[]	Member	left to right	brand[“carName”]
2.2	new	Create		new Date(“July 27, 2023”)
2.3	()	Function Call		myFun()
3	++	Postfix increment	N/A	i++
3.1	—	Postfix Decrement	N/A	i–
4	++	Prefix increment	right to left	++i

4.1	—	Prefix Decrement	N/A	–i
4.2	!	Logical NOT	N/A	!(x===y)
4.3	typeof	Type	N/A	typeof a
5	**	Exponentiation	right to left	4**2
6	*	Multiplication	left to right	2*3
6.1	/	Division		18/9
6.2	%	Remainder		4%2

7	+	Addition	left to right	2+4
7.1	–	Subtraction		4-2
8	<<	Left shift	left to right	y<<2
8.1	>>	Right Shift		y>>2
8.2	>>>	Unsigned right shift		y>>>2
9	<	Less than	left to right	3<4
9.1	<=	Less than or equal to		3<=4

9.2	>	Greater than		4>3
9.3	>=	Greater than or equal to	4>=3	
9.4	In	In		“PI” in MATH
9.5	Instance of	Instance of		A instanceof B
10	==	Equality	left to right	x==y
10.1	!=	Inequality		x!=y
10.2	===	Strictly equal		x===y

10.3	!==	Strictly unequal		x!==y
11	&	Bitwise AND	left to right	x&y
12	^	Bitwise XOR	left to right	x^y
13	|	Bitwise OR	left to right	x|y
14	&&	Logical AND	left to right	x&&y
15	||	Logical OR	left to right	x||y
16	? :	Conditional	right to left	(x>y)?x:y

17	=	Assignment	right to left	x=5
17.1	+=	Addition assignment		x+=5 // x=x+5
17.2	-=	Subtraction assignment		x-=5 // x=x-5
17.3	*=	Multiplication assignment		x*=5 // x = x*5
17.4	/=	Division assignment		x/=5 // x = x/5
17.5	%=	Modulo assignment		x%=5 // x = x%5

17.6	<<=	left shift assignment		x<<=5 // x=x<<5
17.7	>>=	right shift assignment		x>>=5 // x=x>>5
17.8	>>>=	Unsigned right shift assignment		x>>>=5 // x=x>>>5
17.9	&=	Bitwise AND assignment		x&=5 // x=x&5
17.10	^=	Bitwise XOR assignment		x^=5 // x=x^5
17.11	|=	Bitwise OR assignment		x|=5 // x=x|5

18	yield	Pause function	right to left	yield x
19	,	Comma	left to right	x,y

Exception and Error Handling :-

An error is an action that is inaccurate or incorrect. There are three types of errors in programming which are discussed below:

- Syntax error
- Logical error
- Runtime error

Syntax error: According to computer science, a syntax error is an error in the syntax of a sequence of characters or tokens that is intended to be written in a particular programming language or it is also a compile-time error if the syntax is not correct then it will give an error message.

Logical error: It is the most difficult error to be traced as it is the error on the logical part of the coding or logical error is a bug in a program that causes it to operate incorrectly and terminate abnormally (or crash).

Runtime Error: A runtime error is an error that occurs during the running of the program, also known as the exception. In the example that is given below the syntax is correct, but at runtime, it is trying to call a method that does not exist.

As in runtime errors, there are exceptions and these exceptions can be handled with the help of the try-and-catch method.

try-catch method

JavaScript uses the try catch and finally to handle the exception and it also uses the throw operator to handle the exception. try have the main code to run and in the catch block if any error is thrown by try block will be caught and the catch block will execute. Finally block will always occur even if an error is thrown

Note:- We can create our own errors using throw but error thrown can only be String, Number, Boolean, or an object

In JavaScript, errors can be thrown using the throw statement to indicate an exceptional condition. The try block is used to wrap code that might throw an error, and the catch block handles the error, preventing the program from crashing and allowing graceful error management.

But all errors can be solved and to do so we use five statements that will now be explained.

- The **try** statement lets you test a block of code to check for errors.

- The **catch** statement lets you handle the error if any are present.
- The **throw** statement lets you make your own errors.
- The **finally** statement lets you execute code after try and catch.
- The **finally** block runs regardless of the result of the try-catch block

Ex :-

```
function divide(a,b){
    try{
        if(b===0){
            throw new Error("division by zero is not allowed")
        }
        let result = a/b;
        console.log(result); // 0.5
    }
    catch(e){
        console.log(e.message);
    }
    finally{
        console.log(" Execution completed");
    }
```

```
}

divide(20,0);
```

Types of Errors (done by employees)

1, Type Error :-

Ex :- let person=null;

```
console.log(person.name);
```

```
const flowers=new Object({name:null,color:undefined,Family:null});

console.log(flowers.name);
```

2. Logical Error :-

```
let arr = [1, 2, 3, 4, 5];



for (let i = 0; i <= arr.length; i++) {
    try {
        // Attempt to access the array element
        console.log(arr[i]);
    } catch (error) {
```

```
        // Catch any errors  undefined array index
        console.log("An error occurred: " + error.message);
    } finally {
        // This block runs whether an error occurs or not
        console.log("Iteration", i, "complete.");
    }
}
```

Ex :- 2

```
function sumNumbers() {

  //let a=10;
  //let b=15;
  console.log(a + b);  // Logical error: `a` and `b` are used before they are declared
  let a = 10;
  let b = 15;
}

sumNumbers();  // This will throw a ReferenceError
```

Ex :- 3

ex;3// Incorrect Comparisons Error

//"==" "==="

// "!==" "!="

console.log(10=="10") // output is true

console.log(10==="10") // output is false

console.log(10!=="10") // output is true

console.log(10!="10") // output is false

A TypeError is thrown if an operand or argument is incompatible with the

// type expected by an operator or function.

// let num=1;

// console.log(num.toUpperCase()); // we cannot convert a number to upper case

// Accessing a Property on undefined or null

// let person;

// console.log(person.name); //person is undefined, so trying to access its name property results in a TE.

//Calling Something That Is Not a Function

// let greet = "Hello";

// greet(); // greet is a string, not a function, so attempting to call it as a function results in a TypeError.

const b=10;

b=100;

console.log(b); //const we cannot update values so its get typeerror.

Range Error :-

In JavaScript, a `RangeError` is an error that occurs when a value is not within the set or range of allowable values. This typically happens in situations where you're using a number that is outside the permissible range.

Here are a few common scenarios where a `RangeError` might occur:

Invalid Array Length: If you try to create an array with an invalid length (like a negative number or a number that is too large), a `RangeError` will be thrown

```
let arr = new Array(-1); // RangeError: Invalid array length
```

2. Exceeding Maximum Call Stack Size: If you have a recursive function that never terminates, it can cause the call stack to exceed its maximum size, leading to a `RangeError`.

```
Function Func1() {
    Func1();
}
Func(); // Throws RangeError: Maximum call stack size exceeded
```

Internal Error : -

The Internal Error object indicates an error that occurred internally in the JavaScript engine.

Example cases are mostly when something is too large, e.g.:

- **"too many switch cases",**
- **"too many parentheses in regular expression",**
- **"array initializer too large",**
- **"too much recursion".**

Below example illustrate the too much recursion

ex -

```
function test(x) {
    if (x <= 0)
    return;
```

```
    console.log(x);

    test(x-1);  // recursion function()

     }

    test(100000);
```

Note : We will get a Range error here because of browser compatibility (The Error will be shown in Mozilla Firefox browser).

Evaluation Error :-

--> In JavaScript, the EvalError object represents an error regarding the global eval() function. While it's not thrown by the JavaScript engine anymore (since ECMAScript 5), it is still kept for backward compatibility.

--> Typically, you won't encounter EvalError unless you are working with older environments or need to throw it manually in your own code.

Events :-

JavaScript Events are **actions or occurrences** that happen in the browser. They can be triggered by various user interactions or by the browser itself.

Common events include mouse clicks, keyboard presses, page loads, and form submissions. Event handlers are JavaScript functions that respond to these events, allowing developers to create interactive web applications.

Syntax :-

```
<HTML-element Event-Type = "Action to be performed">
```

Common Javascript Events table :-

Event Attribute	Description
onclick	Triggered when an element is clicked.
onmouseover	Fired when the mouse pointer moves over an element.
onmouseout	Occurs when the mouse pointer leaves an element.

onkeydown	Fired when a key is pressed down.
onkeyup	Fired when a key is released.
onchange	Triggered when the value of an input element changes.
onload	Occurs when a page has finished loading.
onsubmit	Fired when a form is submitted.
onfocus	Occurs when an element gets focus.
onblur	Fired when an element loses focus.

Javascript Events Examples :-

Ex :- 1

```
<html>

    <head>

        <script>

            function hi(){

                alert("Hello ");

                console.log("hi function is called on click of event");

            }



         </script>

         <body>

            <button type="button" onclick="hi()" style="margin-left:
50%;">click me event</button>

         </body>

    </head>

    </html>
```

Explanation :-

Here, we will display a message in the alert box when the button is clicked using onClick() event. This HTML document features a button styled to appear in the middle of the page. When clicked, the button triggers the `hi()` JavaScript function, which displays an alert box with the message “hello”.

Event Handlers :-

Event handlers are functions that are executed in response to specific events occurring in the browser.

They can be attached to HTML elements using event attributes like `onclick`, `onmouseover`, etc., or added dynamically using the `addEventListener()` method in JavaScript.

Ex :- 2 (Event Handlers)

```
<html>

<head>

    <title>Event Handler Example</title>

</head>

<body>


<button onclick="myFunction()">Click me</button>
```

```
<script>
    // JavaScript function to handle the click event
    function myFunction() {
        alert("Button clicked!");
    }
</script>
</body>
</html>
```

Ex :- 3 (onmouseover)

```
<html>
 <body>
<script>
function mouseoverevent(){
    alert("mouse over event");
 }
        </script>
<p onmouseover="mouseoverevent()">keep cursor over me</p>
```

```
</body>

</html>
```

Ex :- 3 (onkeydown)

```
<html>

   <body>

    <h2> Enter something here</h2>

    <input type="text" id="text1" onkeydown="keydownevent()">

    <script>

        function keydownevent(){

            document.getElementById("input1");

            alert("Pressed a key");

            console.log("function called");

        }

    </script>

    </body>

</html>
```

Event Bubbling and Event Capturing :-

Event Bubbling and Event Capturing are the two interesting concepts in javascript . Before going deeper into these concepts first let's understand what an **event listener** is ?

An **Event Listener** is basically a function that waits for an event to occur . That event can be anything like mouse click , keyboard press etc . An event listener contains three parameters and it can defined using following syntax

<element>.addEventListener(<eventName>, <callbackFunction>,{capture : Boolean }) ;

- **<element> :-** the element to which the event listener is added .
- **<eventName> :-** it can be “click” , “key up” , “key down” etc .
- **<callbackFunction> :-** This function fires after an event happened .
- **{capture : boolean } :-** it tells whether the event will be in the capture phase or in the bubbling phase (optional) .

Ex :-

```
<html>

    <head>

        <style>

            div {
```

```
    color : white;
    display :flex;
    justify-content:center;
    align-items : center;
    flex-direction: column;
}
#grandparent {
    background-color: green;
    width: 300px;
    height: 300px;
}

#parent {
    background-color: red;
    width: 200px;
    height: 200px;
}
#child {
    background-color: blue;
```

```
            width: 100px;
            height: 100px;
        }
    </style>
</head>
<body>
<div>
    <div id="grandparent">GrandParent
        <div id="parent">
            Parent
        <div id="child">
            Child
        </div>
        </div>
    </div>
</div>
<script>
const grandParent= document.getElementById("grandparent");
const Parent= document.getElementById("parent");
```

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



    grandParent.addEventListener("click",(e)=>{
     console.log("GrandParent");
    }, {capture :false});



    Parent.addEventListener("click",(e)=>{
     console.log("Parent");
    }, {capture :false});



    child.addEventListener("click",(e)=>{
     console.log("Child");
    }, {capture :false});
    </script>
     </body>
</html>
```

Conclusion :-

Event bubbling in JavaScript allows events to propagate from the innermost to the outermost elements, enabling multiple event listeners to handle the same

event in a hierarchical order. This ensures that both specific and general event handling can be effectively managed.

Ex :- 2

```
<!DOCTYPE html>

<html>

<head>



    <style>

        div {

            color: white;

            display: flex;

            justify-content: center;

            align-items: center;

            flex-direction: column;

        }

        h2 {

            color: black;

        }

        #grandparent {
```

```
            background-color: green;
            width: 300px;
            height: 300px;
        }
        #parent {
            background-color: blue;
            width: 200px;
            height: 200px;
        }
        #child {
            background-color: red;
            width: 100px;
            height: 100px;
        }
    </style>
</head>

<body>
    <div>
```

```
    <h2>Welcome To GFG</h2>
    <div id="grandparent">GrandParent
        <div id="parent">Parent
            <div id="child"> Child</div>
        </div>
    </div>
</div>
<script>
    const grandParent = document.getElementById("grandparent");
    const parent = document.getElementById("parent");
    const child = document.getElementById("child");

    // Changing value of capture parameter as 'true'
    grandParent.addEventListener("click", (e) => {
        console.log("GrandParent");
    }, { capture: true });
    parent.addEventListener("click", (e) => {
        console.log("Parent");
    }, { capture: true });
```

```
        child.addEventListener("click", (e) => {
            console.log("Child");
        }, { capture: true });
    </script>
</body>


</html>
```

It's clearly visible that the *ancestor divs* of the *child div* were printing first and then the *child div* itself. So, the process of propagating from the farthest element to the closest element in the DOM is called event capturing**.** Both terms are just opposite of each other.

If we clicked on the child *div*, the propagation is stopped on *parent div* and does not move to the grandparent *div*. Hence, the event bubbling is prevented.

Note: The event capturing can also be prevented using the same way.

Important points to remember:

- If we do not mention any third parameter in **addEventListener()**, then by default event bubbling will happen.

- Event bubbling and event capturing happen only when the element and its ancestors have the same event listener (in our case, ‘click’ event) attach to them.

Ex :- 3

```
<!DOCTYPE html>

<html>



<head>

    <style>

        div {

            color: white;

            display: flex;

            justify-content: center;

            align-items: center;

            flex-direction: column;

        }

        h2 {

            color: black;

        }
```

```
        #grandparent {
            background-color: green;
            width: 300px;
            height: 300px;
        }
        #parent {
            background-color: blue;
            width: 200px;
            height: 200px;
        }
        #child {
            background-color: red;
            width: 100px;
            height: 100px;
        }
    </style>
</head>

<body>
```

```
<div>
    <div id="grandparent">GrandParent
        <div id="parent">Parent
            <div id="child"> Child</div>
        </div>
    </div>
</div>

<script>
    const grandParent = document.getElementById("grandparent");
    const parent = document.getElementById("parent");
    const child = document.getElementById("child");
    grandParent.addEventListener("click", (e) => {
        console.log("GrandParent bubbling");
    });
    parent.addEventListener("click", (e) => {
        e.stopPropagation(); //syntax to stop event bubbling
        console.log("Parent bubbling");
    });
```

```
            child.addEventListener("click", (e) => {
                e.stopPropagation(); //syntax to stop event bubbling
                console.log("Child bubbling");
            });
        </script>
    </body>


    </html>
```

We can stop event bubbling and event capturing by using stopPropagation() method .

Conclusion: We have learned about event bubbling and event capturing and these are some key points.

- Event capturing means propagation of event is done from ancestor elements to child element in the DOM while event bubbling means propagation is done from child element to ancestor elements in the DOM.
- The event capturing occurs followed by event bubbling.
- If {capture: true} ,event capturing will occur else event bubbling will occur.

- Both can be prevented by using the **stopPropagation()** method.

Number Properties and it's Methods :-

- **toString() :-** it returns a number as a string
- **toExponential :-** toExponential() returns a string, with a number rounded and written using exponential notation.
- **toFixed() :-** toFixed() returns a string, with the number written with a specified number of decimals:
- **toPrecision() :-** toPrecision() returns a string, with a number written with a specified length:
- **valueOf() :-** returns number as number

In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object).

The valueOf() method is used internally in JavaScript to convert Number objects to primitive values.

There is no reason to use it in your code.

Ex :- 1

```
let x=123;

console.log(typeof(x)); // number

let numString=x.toString();

console.log(typeof(numString)); // string
```

```
let y=1234.653;

console.log(y.toExponential());

 console.log(y.toFixed(2));

 console.log(y.toPrecision(7));
```

Converting Variables to Numbers :-

- **Number() :-** The Number() method can be used to convert JavaScript variables to numbers, If given value cannot be converted to a number it gives NaN as output .
- **parseInt() :-** parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned, If the number cannot be converted, NaN (Not a Number) is returned.

Properties of Number :-

EPSILON	The difference between 1 and the smallest number > 1.
MAX_VALUE	The largest number possible in JavaScript

MIN_VALUE	The smallest number possible in JavaScript
MAX_SAFE_INTEGER	The maximum safe integer ($2^{53} - 1$)
MIN_SAFE_INTEGER	The minimum safe integer $-(2^{53} - 1)$
POSITIVE_INFINITY	Infinity (returned on overflow)
NEGATIVE_INFINITY	Negative infinity (returned on overflow)
NaN	A "Not-a-Number" value

Ex :-

```
console.log(Number.MAX_VALUE);

console.log(Number.MIN_VALUE);

console.log(Number.NaN);

console.log(Number.POSITIVE_INFINITY);

console.log(Number.NEGATIVE_INFINITY);

console.log(Number.EPSILON);
```

```
console.log(Number.isInteger(123.89));

console.log(Number.isFinite(Infinity));
```

Important Terms :-

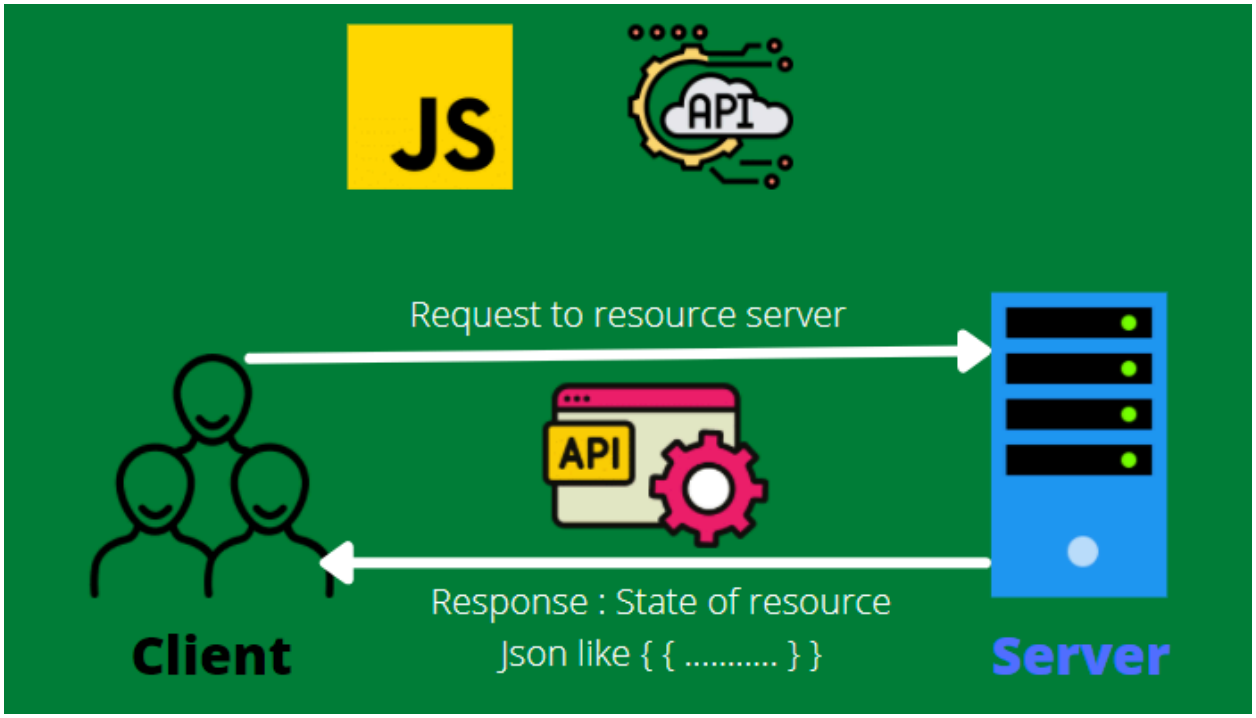
1. **API :-** Application Programming interface is a set of defined rules that enables different applications , services or systems to communicate with each other. It allows one system to request services or data from another system and receive responses in a structured format .

APIs are commonly used to fetch data (from third party or server) , Send data (to server or third party , Interact with services (authentication ,payment gateway , or social media platforms) .

Types of APIs :-

1. **Web API :-** A web API is an api that is accessed via web , typically using HTTP requests .
2. **LIbrary API :-** A library API is an api that is provided by Javascript library or framework . (Ex :- jQuery API) .
3. **Operating System API :-** An OS API is an api that allows Javascript to interact with the underlying OS . (Nodejs FS file system api) .

API Request / Response :-



1. Request :- A request is sent from the client-side to the server-side.

2. Response :- The server-side sends a response back to the client.

Common API Request Methods :-

1. **GET :-** Retrieve data
2. **POST :-** Send data
3. **PUT :-** Update data
4. **DELETE :-** Delete data

API Response Formats :-

1. **JSON :-** (Javascript object notation) A lightweight data interchange format .

2. **XML :-** (Extensible markup Language) A markup language for structuring data .

In Javascript you can work with APIs using :

1. **Fetch API :-** A modern API for making HTTP requests .
2. **XMLHttpRequest :-** An older API for making HTTP requests .
3. **Libraries like Axios :-** A popular library for making HTTP requests .

By using APIs , Javascript can interact with different systems , services and data sources , enabling the creation of powerful and dynamic applications .

Web Storage :-

Web storage in Javascript refers to the ability to store data locally within a user's web browser. It provides a way to store and retrieve data without relying on server-side storage or cookies .

What is Web Storage API ?

The web storage API in JavaScript allows us to store the data in the user's local system or hard disk. Before the storage API was introduced in JavaScript, cookies were used to store the data in the user's browser.

The main problem with the cookies is that whenever browsers request the data, the server must send it and store it in the browser. Sometimes, it can also happen that attackers can attack and steal the data.

In the case of the storage API, we can store the user's data in the browsers, which remains limited to the user's device.

JavaScript contains two different objects to store the data in the local.

1. **Local storage**
2. **Session storage**

The window local storage object :-

The localStorage object allows you to store the data locally in the key-value pair format in the user's browser.

You can store a maximum of 5 MB data in the local storage.

Whatever data you store into the local storage, it never expires. However, you can use the removeItem() method to remove the particular or clear() to remove all items from the local storage.

Syntax :-

```
localStorage.setItem(key, value); // To set key-value pair

localStorage.getItem(key); // To get data using a key
```

Characteristics of local storage :-

1. **Client-side storage :-** Data is stored locally on the client-side (users browser) .
2. **No expiration date :-** Data is stored until explicitly deleted .
3. **Storage capacity :-** Typically 5MB per domain .
4. **Security :-** Data is stored in a sandboxed environment , reducing the risk of data breaches .
5. **Access :-** Data can be accessed using javascript methods .

Local storage API :-

- **localStorage :-** Accesses the local storage .
- **setItem(key , value) :-** stores a value .
- **getItem(key) :-** retrieves a value .
- **removeItem(key) :-** deletes a value .
- **clear() :-** deletes all values.
- **index :-** an integer representing an index . (starters from 0) .
- **length :-** returns the number of key value pairs .

Ex :-

```
<!DOCTYPE html>

<html>

<head>

</head>

<body>

    <h1>Local Storage</h1>

    <script>

        localStorage.setItem('username',"Rahul");

        localStorage.setItem('password',"Rahul123");



        console.log(localStorage.getItem('username'));

        console.log(localStorage.getItem('password'));

         localStorage.removeItem('username');
```

```
        localStorage.clear();
    </script>
</body>
</html>
```

Use cases for Local Storage :-

- **User Preferences -** Store user settings , such as theme or language preferences .
- **Cache -** Stores frequently-used data to reduce server requests .
- **Offline Support -** Store data locally for offline access .

Note :- Local storage is not suitable for sensitive data , as it can be accessed by any script on the same domain .

Parameters :-

Key :- can be a string

Value :- It is the value in string format

Ex :- (**storing an object in local storage)**

```
<html>
    <body>
        <button type="button" onclick="setItem()">SetItem</button> <br>
<br>
```

```
        <button type="button" onclick="getItem()">GetItem</button> <br>
<br>

        <button type="button" onclick="removeItem()">RemoveItem</button>
<br> <br>

        <button type="button"
onclick="removeProperty()">RemoveProperty</button> <br> <br>



        <p id="demo"></p>

        <script>

            const output=document.getElementById("demo");

            function setItem(){

                const person={

                    Fname:"John",

                    Lname:"Doe",

                    Address:"HYD"

                }

                localStorage.setItem('person',JSON.stringify(person));

            }

            function getItem(){

               const person= localStorage.getItem('person');

               console.log(typeof(person));

               output.innerHTML="Details of a Person :" + person;

            }
```

```
            function removeProperty(){

              const storedObject= JSON.parse
(localStorage.getItem('person'));

                // Remove  the 'address property'

                console.log(typeof(storedObject));

                delete storedObject.Address;

                // update the local storage


localStorage.setItem('person',JSON.stringify(storedObject));

                localStorage.getItem('person');

                if(storedObject!==null&& storedObject!==undefined){

                    console.log(typeof JSON.stringify(storedObject)); //
string



                }

            }

            // removing an object from local storage

            function removeItem(){

                localStorage.removeItem('person');

                output.innerHTML="Remove the object from local stoarge";

            }

            console.log(localStorage.length);
```

```
        </script>

    </body>

</html>
```

JSON.stringify() :- It is a method that converts a javascript object or value to a JSON string .

JSON.parse() :- It is a method that parses a JSON string and returns Javascript object

You can check the type of the value that you have stored in local storage it will return you a string , when you parse it back , it will return the output as object .

The Window sessionStorage Object

The sessionStorage object also allows storing the data in the browser in the key-value pair format.

It also allows you to store the data up to 5 MB.

The data stored in the session storage expires when you close the tab of the browsers. This is the main difference between the session storage and local storage. You can also use the removeItem() or clear() method to remove the items from the session storage without closing the browser's tab.

Note – Some browsers like Chrome and Firefox maintain the session storage data if you reopen the browser's tab after closing it. If you close the browser window, it will definitely delete the session storage data.

Syntax

Follow the syntax below to use the session storage object to set and get data from the session storage.

```
sessionStorage.setItem(key, value); // To set key-value pair

sessionStorage.getItem(key); // To get data using a key
```

Characteristics of session storage :-

1. **Temporary Storage :-** Data stored in SessionStorage is deleted when the user closes the browser.
2. **Origin-based :-** Data is stored and accessible only from the same origin (domain , protocol , port) .
3. **Key - value pairs :-** Data is stored as key-value pairs .
4. **Limited storage :-** Limited up to 5MB .
5. **Browser-based :-** Data is stored locally within the browser.
6. **Access :-** Data is accessed and manipulated using Javascript .

Scope :- Data is scoped to the current browser session , to the same origin , protocol , domain .

Session storage API :-

- **sessionStorage :-** Accesses the session storage .
- **setItem(key , value) :-** stores a value .
- **getItem(key) :-** retrieves a value .
- **removeItem(key) :-** deletes a value .
- **clear() :-** deletes all values.
- **length :-** returns the number of key value pairs .

- **index :-** an integer representing an index . (starters from 0)

Ex:-

```
<html>

<body>

   <button onclick = "setItem()"> Set Item </button>

   <button onclick = "getItem()"> Get Item </button>

   <p id = "output"> </p>

   <script>

      const output = document.getElementById('output');

      function setItem() {

         sessionStorage.setItem("username", "React");

      }

      function getItem() {

         const username = sessionStorage.getItem("username");

         output.innerHTML = "The user name is: " + username;

     }

   </script>

</body>

</html>
```

Validations :- Validation is a process of checking user input data to ensure it meets specific requirements , such as format , length , or type .

Purpose :-

1. Prevents Errors and Exceptions.
2. Ensure data consistency and accuracy .
3. Protect against security threats (cross-site attacks) .
4. Improve user experience .

Types of Form Validation

Client-side Validation: - This is done in the user's browser before the form is submitted. It provides quick feedback to the user, helping them fix errors without sending data to the server first.

Server-side Validation: - Even though client-side validation is useful, it's important to check the data again on the server. This ensures that the data is correct, even if someone tries to bypass the validation in the browser.

Javascript Form Validation :-

JavaScript Form Validation is a way to ensure that the data users enter into a form is correct before it gets submitted. This helps ensure that things like emails, passwords, and other important details are entered properly, making the user experience smoother and the data more accurate.

Steps for form validation in Javascript :-

When we validate a form in JavaScript, we typically follow these steps:

Data Retrieval :- The first step is to get the user's values entered into the form fields (like name, email, password, etc.). This is done using document.forms.RegForm, which refers to the form with the name "RegForm".

Data Validation :-

- Name Validation: We check to make sure the name field isn't empty and doesn't contain any numbers.
- Address Validation: We check that the address field isn't empty.
- Email Validation: We make sure that the email field isn't empty and that it includes the "@" symbol.
- Password Validation: We ensure that the password field isn't empty and that the password is at least 6 characters long.
- Course Selection Validation: We check that a course has been selected from a dropdown list.

  Error Handling :- If any of the checks fail, an alert message is shown to the user using window.alert, telling them what's wrong.The form

focuses on the field that needs attention, helping the user easily fix the error.

Submission Control :- If all the validation checks pass, the function returns true, meaning the form can be submitted. If not, it returns false, stopping the form from being submitted.

Focus Adjustment :-The form automatically focuses on the first field that has - an error, guiding the user to fix it.

Ex :-

Html :-

```
<html>
    <head>
        <title>Form Validation</title>
        <link rel="stylesheet" href="style.css">
    </head>
    <body>
        <h1>Registraion Form</h1>
        <form name="Regform" onsubmit="return validateForm()">
            <p>
                <label for="name">Name: </label>
```

```
                <input type="text" id="name" name="Name"
placeholder="Enter your fullname"/>

                <span id="name-error" class="error-message"></span>

            </p>



            <p>

                <label for="address">Address: </label>

                <input type="text" id="address" name="Address"
placeholder="Enter your full address"/>

                <span id="address-error" class="error-message"></span>

            </p>



            <p>

                <label for="email">Email-id: </label>

                <input type="text" id="email" name="Email"
placeholder="Enter your email"/>

                <span id="email-error" class="error-message"></span>

            </p>



            <p>

                <label for="password">Password: </label>

                <input type="password" id="password" name="Password"
placeholder="Enter your password"/>
```

```
        <span id="password-error" class="error-message"></span>
    </p>

    <p>
        <label for="subject">Select Your Course:</label>
        <select id="subject" name="Subject">
            <option value="">
                Select Course
            </option>
            <option value="BTECH">
                BTECH
            </option>
            <option value="BBA">
                BBA
            </option>
            <option value="BCA">
                BCA
            </option>
            <option value="B.COM">
                B.COM
            </option>
        </select>
```

```
            <span id="subject-error" class="error-message"></span>
        </p>

        <p>
            <label for="comment">College Name:</label>
            <textarea id="comment" name="Comment"></textarea>
        </p>
        <p>
            <input type="checkbox" id="agree" name="Agree" />
            <label for="agree">I agree to the above
                information</label>
            <span id="agree-error" class="error-message"></span>
        </p>

        <p>
            <input type="submit" value="Send" name="Submit" />
            <input type="reset" value="Reset" name="Reset" />
        </p>
    </form>
    <script src="script.js"></script>

</body>
```

```
</html>
```

CSS :-

```
body {
    font-family: Arial, sans-serif;
    background-color: #f5f5f5;
}

h1 {
    text-align: center;
    color: #333;
}

form {
    max-width: 600px;
    margin: 0 auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

input[type="text"],
```

```
input[type="password"],

select,

textarea {

    width: 100%;

    padding: 10px;

    margin: 5px 0;

    border: 1px solid #ccc;

    border-radius: 5px;

    box-sizing: border-box;

    font-size: 16px;

}



select {

    width: 100%;

    padding: 10px;

    margin: 5px 0;

    border: 1px solid #ccc;

    border-radius: 5px;

    box-sizing: border-box;

    font-size: 16px;

    background-color: #fff;

    appearance: none;
```

```
    -webkit-appearance: none;
    -moz-appearance: none;
}

textarea {
    resize: vertical;
}

input[type="submit"],
input[type="reset"],
input[type="checkbox"] {
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 5px;
    padding: 10px 20px;
    cursor: pointer;
    font-size: 16px;
}

input[type="submit"]:hover,
input[type="reset"]:hover,
```

```
input[type="checkbox"]:hover {
    background-color: #0056b3;
}

.error-message {
    color: red;
    font-size: 14px;
    margin-top: 5px;
}
```

script.js :-

```
function validateForm() {
    const name = document.getElementById("name").value;
    const address = document.getElementById("address").value;
    const email = document.getElementById("email").value;
    const password = document.getElementById("password").value;
    const subject = document.getElementById("subject").value;
    const agree = document.getElementById("agree").checked;

    const nameError = document.getElementById("name-error");

    const addressError = document.getElementById("address-error");
```

```
const emailError = document.getElementById("email-error");

const passwordError = document.getElementById("password-error");

const subjectError = document.getElementById("subject-error" );

const agreeError = document.getElementById("agree-error");


nameError.textContent = "";

addressError.textContent = "";

emailError.textContent = "";

passwordError.textContent = "";

subjectError.textContent = "";

agreeError.textContent = "";


let isValid = true;


if (name === "" || /\d/.test(name)) {

    nameError.textContent =

        "Please enter your name properly.";

    isValid = false;

}


if (address === "") {

    addressError.textContent =
```

```
        "Please enter your address.";
    isValid = false;
}

if (email === "" || !email.includes("@")) {
    emailError.textContent =
        "Please enter a valid email address.";
    isValid = false;
}

if (password === "" || password.length < 6) {
    passwordError.textContent =
        "Please enter a password with at least 6 characters.";
    isValid = false;
}

if (subject === "") {
    subjectError.textContent =
        "Please select your course.";
    isValid = false;
}
```

```
    if (!agree) {
        agreeError.textContent =
            "Please agree to the above information.";
        isValid = false;
    }

    return isValid;
}
```

Regular Expressions :-

In Javascript , Regular Expression (regex) is a pattern used to match and manipulate the text .

A regex is a string of characters that defines a search pattern

Syntax:- Regex patterns are enclosed within slashes (/) or can created using the RegExp object .

/ literal pattern / flag

Or

new RegExp(‘pattern’,’flag’)

A regular expression is a character sequence defining a search pattern. It’s employed in text searches and replacements, describing what to search for

within a text. Ranging from single characters to complex patterns, regular expressions enable various text operations with versatility and precision.

A regular expression can be a single character or a more complicated pattern.

Explanation :-

/GeeksforGeeks/i is a regular expression.

GeeksforGeeks is the pattern (to be used in a search).

i is a modifier (modifies the search to be Case-Insensitive).

In JavaScript, regular expressions are objects. JavaScript provides the built-in RegExp type that allows you to work with regular expressions effectively.

Regular expressions are useful for searching and replacing strings that match a pattern. They have many useful applications.

Creating a Regular Expression :-

To create a regular expression in JavaScript, you enclose its pattern in forward-slash characters (/) like this:

let re = /hi/

Or you can use the RegExp constructor:

Let re=new RegEx('hi);

Both regular expressions are instances of the `RegExp` type. They match the string `'hi'`.

Testing for matching :-

The `RegExp` object has many useful methods. One of them is the `test()` method that allows you to test if a string contains a match of the pattern in the regular expression.

The `test()` method returns `true` if the string argument contains a match.

The `test()` method returns `true` if the string argument contains a match.

The following example use the `test()` method to test whether the string `'hi John'` matches the pattern `hi` :

Ex :-

```
let re = /hi/;

let result = re.test('hi John');

console.log(result); // true
```

Modifiers :-

Modifiers are also called flags , they are used to change the behavior of the pattern . Common modifiers that are used along with patterns are as follows ,they can be used to perform multi line searches which can also be set to case-insensitive matching:

Expressions	Descriptions
g	Find the character globally
i	Find a character with case-insensitive matching
m	Find multiline matching

Brackets :-

It finds the characters in a specified range .

Expressions	Description

[abc]	Find any of the characters inside the brackets
[^abc]	Find any character, not inside the brackets
[0-9]	Find any of the digits between the brackets 0 to 9
[^0-9]	Find any digit not in between the brackets
(x | y)	Find any of the alternatives between x or y separated with |

Meta characters :-

They are characters with special meaning .

Metacharacter	Description
\.	Search single characters, except line terminator or newline.
\w	Find the word character i.e. characters from a to z, A to Z, 0 to 9
\d	Find a digit
\D	Search non-digit characters i.e all the characters except digits
\s	Find a whitespace character

\S	Find the non-whitespace characters.
\b	Find a match at the beginning or at the end of a word
\B	Find a match that is not present at the beginning or end of a word.
\0	Find the NULL character.
\n	Find the newline character.
\f	Find the form feed character
\r	Find the carriage return character

\t	Find the tab character
\v	Find the vertical tab character
\uxxxx	Find the Unicode character specified by the hexadecimal number xxx

Quantifiers :-

Quantifiers in regular expressions (regex) specify how many times a character or group of characters can occur.It defines quantities occurrence .

‘

Quantifier	Description
n+	Match any string that contains at least one n

n*	Match any string that contains zero or more occurrences of n
n?	Match any string that contains zero or one occurrence of n
m{X}	Find the match of any string that contains a sequence of m, X times
m{X, Y}	Find the match of any string that contains a sequence of m, X to Y times
m{X,}	Find the match of any string that contains a sequence of m, at least X times

m$	Find the match of any string which contains m at the end of it
^m	Find the match of any string which contains m at the beginning of it
?!m	Find the match of any string which is not followed by a specific string m.

Regular Expression Object Properties:

Property	Description
constructor	Return the function that created the RegExp object's prototype

global	Specify whether the **“g”** modifier is set or not
ignorecase	Specify whether the **“i”** modifier is set or not
lastindex	Specify the index at which to start the next match
multiline	Specify whether the **“m”** modifier is set or not
source	Return the text of RegExp pattern

Ex:- (Modifiers)

```
let re = /HI/i;

let result = re.test('hi John');


console.log(result); // true
```

```
console.log(typeof(re));


let regex2=/hello/g;

let string='hello hi how r u hello';

console.log(string.match(regex2));


const reg=/^Hello/mig;

const string2="hello world\nhello again";

console.log(string2);
```

Reference (examples)

```
[^abc] --> negated classes , a , b , c

[a-z]  --> range


- :- used define range

^ :- negates the class when u use inside brackets

\ :-  use escape special chars

\d :-  matches a digit

\D :- matches non - digits

\w :- word chars ( letters , digits , underscores )

\W :- non word characters

\s :- matches whitespaces

\S :- non white spaces
```

. :- letter , number , symbol , it acts as placeholders

Ex :-

1. match any digit :- [\d] or [0-9]

2. match any letter :- [a-zA-Z]

3.match alphamuneric :- [a-zA-Z0-9]

4. match non-alphanumeric :- [^a-zA-Z0-9]

5. match specific set :- [abc]

my name is john - text

is - search pattern

m+ :- alteast once (more than one also)

[a-z]+ :- finds any character , atleast once

[0-9]+ :- matches any number , atleast once

[0-9]* :- zero or more

[0-9]{3} :- matches any digits , 3 times exactly

[0-9]{3,} :- matces any digit , max can be any count

^[0-9]$:- beginning and end of line

Ex :- (without regex)

Find occurrence of p from following string

```
var str= "Happy Holi to everyone ";

function occurence(str,char){

    var strarray=str.split('');

    console.log(strarray);


    var result=strarray.filter( function(value){

        return value == char;

    })

    console.log(result);

}

occurence(str,'p');
```

Ex :- 2 (same example using regex)

```
var str= "HapPy holi to everyone";

var pattern=/p/gi;

console.log(pattern.test(str));

console.log(str.match(pattern).length);



var s="xcy";

var pattern=/x[abc]y/;

console.log(pattern.test(s));
```

```
var text="ABD879";

var pattern=/^[A-Z]/;

console.log(pattern.test(text));



var text="123ABD";

var pattern=/[^A-Z]/;

console.log(pattern.test(text));



var mail="vinay@gmail.com";

var pattern=/@gmail.com/;

console.log(pattern.test(mail));



var str="$#%^&a";

var pattern=/\w/;

console.log(pattern.test(str));

var str="ASDE1@#";

var pattern= /\d/;

console.log(pattern.test(str));



var str="1234";

var pattern= /\D/;

console.log(pattern.test(str));
```

```
var str= "vinay12ASW";

var pattern=/[a-z][0-9]+/;

console.log(pattern.test(str));

var str= "x1y";

var pattern=/x[0-9]?y/;

console.log(pattern.test(str));

var str= "x1256y";

var pattern=/x[0-9]{4}y/;

console.log(pattern.test(str));

var str= "x12568878y";

var pattern=/x[0-9]{4,}y/;

console.log(pattern.test(str));

var str= "HYD22MPDP";

var pattern=/^HYD[0-9]{1,3}MPDP$/;

console.log(pattern.test(str));
```