DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
S. R. K. R ENGINEERING COLLEGE BHIMAVARAM

CERTIFICATE

This is to certify that the bonafide work on **Full Stack Development Laboratory** has been submitted by **Mr. Arumilli Vijay Babu (23B91A6109)** as part of the Full Stack Development Laboratory report. This submission is made in partial fulfillment of the requirements for the award of the **Degree of Bachelor of Technology in Computer Science and Engineering** during the **academic year 2024-2025**.

The candidate has successfully carried out this work under my **supervision and guidance**, adhering to the prescribed academic standards and guidelines.

Signature of Lab Incharge **Total Marks:**

INDEX

LAB PROGRAMMES

Exercise-1.HTML Lists, Links and Images

a. Write a HTML program, to explain the working of lists.

Note**:** It should have an ordered list, unordered list, nested lists and ordered list in an Unordered list and definition lists.

INTRODUCTION:

HTML (HyperText Markup Language) is the fundamental building block for web development, enabling the creation of structured and organized content on web pages. One of the essential components of HTML is the use of lists, which help in presenting information in an easily readable format.

Lists in HTML can be categorized into different types: ordered lists, unordered lists, nested lists, and definition lists. Ordered lists display items in a sequential manner using numbers, while unordered lists represent items with bullet points. Additionally, lists can be nested to create a hierarchical structure, and definition lists allow for the representation of terms and their descriptions.

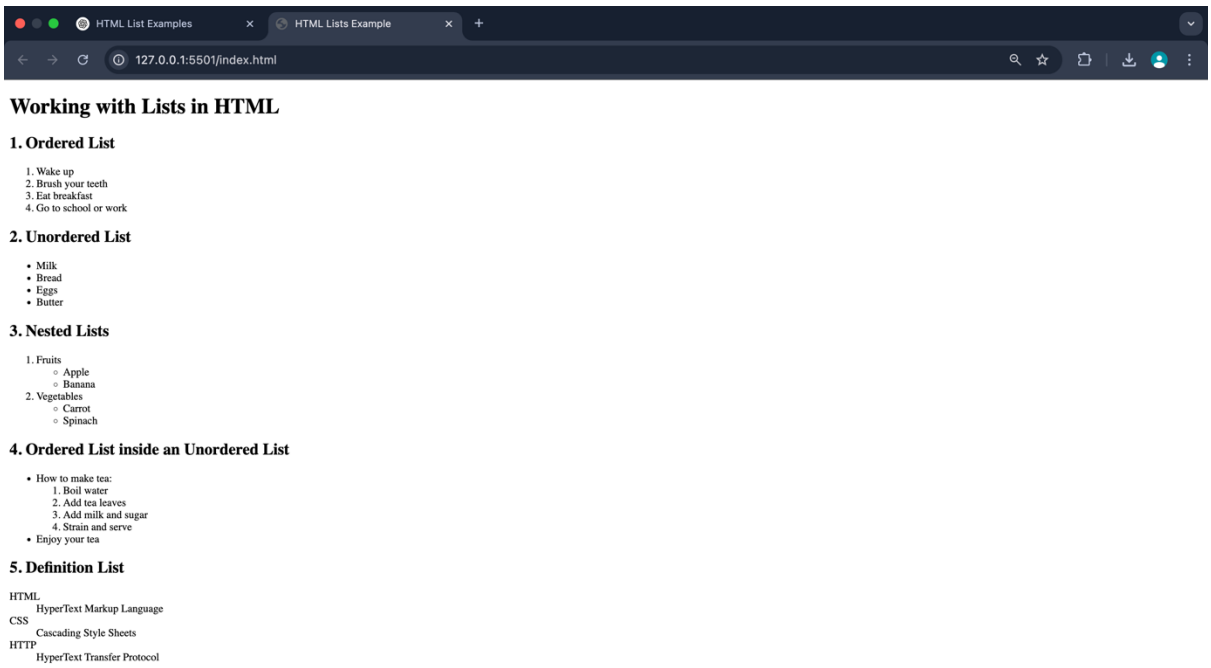
This program effectively demonstrates the implementation and working of various types of lists in HTML, illustrating their significance in content arrangement and user readability. The correct usage of lists enhances web page organization and provides a structured way to present data.

PROGRAMME:

```
<!DOCTYPE html>
<html>
<head>
        <title>HTML Lists Example</title>
</head>
<body>

        <h1>Working with Lists in HTML</h1>

        <!-- Ordered List -->
        <h2>1. Ordered List</h2>
        <ol>
                <li>Wake up</li>
                <li>Brush your teeth</li>
                <li>Eat breakfast</li>
                <li>Go to school or work</li>
        </ol>

        <!-- Unordered List -->
        <h2>2. Unordered List</h2>
        <ul>
                <li>Milk</li>
                <li>Bread</li>
                <li>Eggs</li>
                <li>Butter</li>
        </ul>

 <!-- Nested Lists -->
```

```
<h2>3. Nested Lists</h2>
<ol>
        <li>Fruits
                <ul>
                        <li>Apple</li>
                        <li>Banana</li>
                </ul>
        </li>
        <li>Vegetables
                <ul>
                        <li>Carrot</li>
                        <li>Spinach</li>
                </ul>
        </li>
</ol>

<!-- Ordered List inside Unordered List -->
<h2>4. Ordered List inside an Unordered List</h2>
<ul>
        <li>How to make tea:
                <ol>
                        <li>Boil water</li>
                        <li>Add tea leaves</li>
                        <li>Add milk and sugar</li>
                        <li>Strain and serve</li>
                </ol>
        </li>
        <li>Enjoy your tea</li>
</ul>

<!-- Definition List -->
<h2>5. Definition List</h2>
<dl>
  <dt>HTML</dt>
  <dd>HyperText Markup Language</dd>

  <dt>CSS</dt>
  <dd>Cascading Style Sheets</dd>

  <dt>HTTP</dt>
  <dd>HyperText Transfer Protocol</dd>
</dl>

</body>
</html>
```

OUTPUT:



b. Write a HTML program, to explain the working of hyperlinks using<a> tag and href, target attributes.

Introduction

Hyperlinks are a fundamental feature of web development, allowing users to navigate between web pages, external resources, and specific sections within a document. In HTML, the (anchor) tag is used to create hyperlinks, and it is accompanied by attributes such as and to define the destination and behaviour of the link.

The attribute specifies the URL or location that the hyperlink will direct to, while the attribute determines how the link will open—whether in the same window, a new tab, or within a specific frame. These attributes enhance user experience by enabling seamless navigation between different sections of a website or external pages.

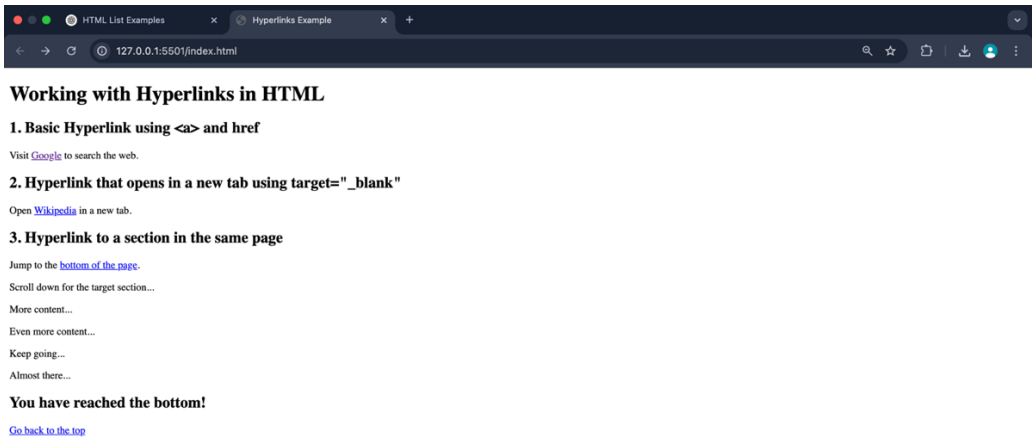
This program illustrates the use of hyperlinks in HTML and showcases various ways in which they can be implemented to improve web page accessibility and interactivity.

Programme:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hyperlinks Example</title>
</head>
<body>

  <h1>Working with Hyperlinks in HTML</h1>
```

```
<!-- Basic hyperlink -->
<h2>1. Basic Hyperlink using &lt;a&gt; and href</h2>
<p>
  Visit <a href="https://www.google.com">Google</a> to search the web.
</p>

<!-- Hyperlink with target="_blank" -->
<h2>2. Hyperlink that opens in a new tab using target="_blank"</h2>
<p>
  Open <a href="https://www.wikipedia.org" target="_blank">Wikipedia</a> in a new tab.
</p>

<!-- Hyperlink to a section within the same page -->
<h2>3. Hyperlink to a section in the same page</h2>
<p>
  Jump to the <a href="#bottom">bottom of the page</a>.
</p>

<!-- Dummy content to scroll -->
<p>Scroll down for the target section...</p>
<p>More content...</p>
<p>Even more content...</p>
<p>Keep going...</p>
<p>Almost there...</p>

<!-- Target section -->
<h2 id="bottom">You have reached the bottom!</h2>
<p><a href="#top">Go back to the top</a></p>

</body>
</html>
```

OUTPUT:



C.Create a HTML document that has your image and your friend's image with a specific height and width. Also when clicked on the images it should navigate to their respective profiles.

Introduction

Images play a crucial role in enhancing the visual appeal and interactivity of web pages. In HTML, the tag is used to display images, and attributes such as and help in controlling their dimensions. Additionally, hyperlinks can be embedded within images using the tag to navigate to specific web pages or user profiles when clicked.

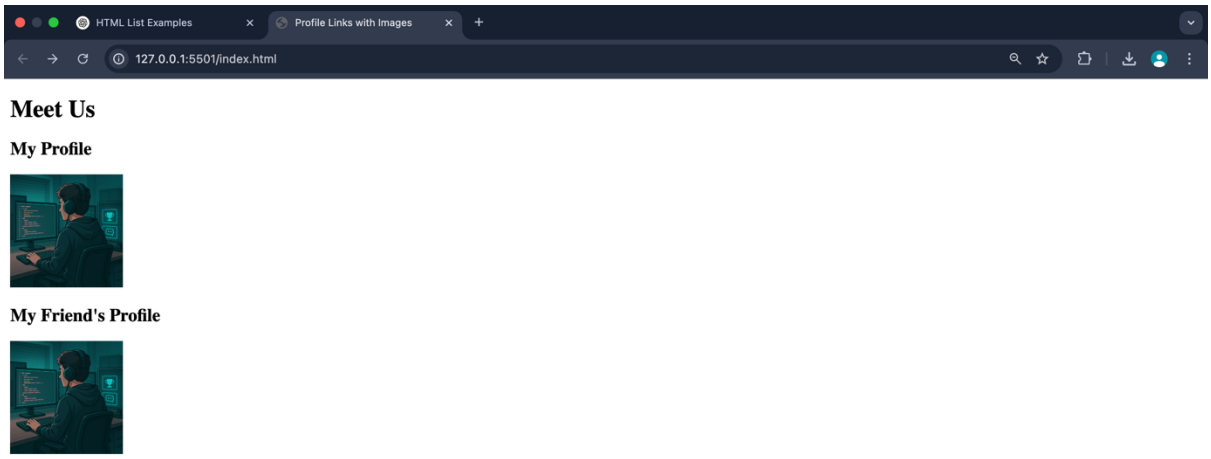
By combining images with hyperlinks, web developers can create interactive elements that guide users efficiently to relevant destinations. This technique is widely used in portfolio websites, social media links, and other online platforms to establish seamless navigation.

This program demonstrates the use of images in HTML with predefined dimensions and interactive functionality, allowing users to click on images to access respective profiles.

Programme:

```
<!DOCTYPE html>
<html>
<head>
  <title>Profile Links with Images</title>
</head>
<body>

  <h1>Meet Us</h1>

  <!-- Your Profile -->
  <h2>My Profile</h2>
  <a href="https://www.linkedin.com/in/your-profile" target="_blank">
    <img src="https://via.placeholder.com/150x150.png?text=Me" alt="My Image"
width="150" height="150">
  </a>

  <!-- Friend's Profile -->
  <h2>My Friend's Profile</h2>
  <a href="https://www.linkedin.com/in/friend-profile" target="_blank">
    <img src="https://via.placeholder.com/150x150.png?text=Friend" alt="Friend's Image"
width="150" height="150">
  </a>

</body>
</html>
```

OUTPUT:



D. Write a HTML program, in such a way that, rather than placing large images on a page, the preferred technique is to use thumbnails by setting the height and width parameters to something like to 100*100 pixels. Each thumbnail image is also a link to a full sized version of the image. Create an image gallery using this technique

Introduction

Creating an image gallery using thumbnails is an efficient technique that enhances webpage performance and improves user experience. Instead of displaying large images directly on a webpage, thumbnails are used to provide a compact preview, reducing load time and maintaining a clean layout.

In HTML, the tag is used to define images, and attributes such as and allow developers to resize them to thumbnail dimensions (e.g., 100×100 pixels). Additionally, the tag is used to link each thumbnail to its corresponding full-sized image, ensuring seamless navigation when a user clicks on the thumbnail.

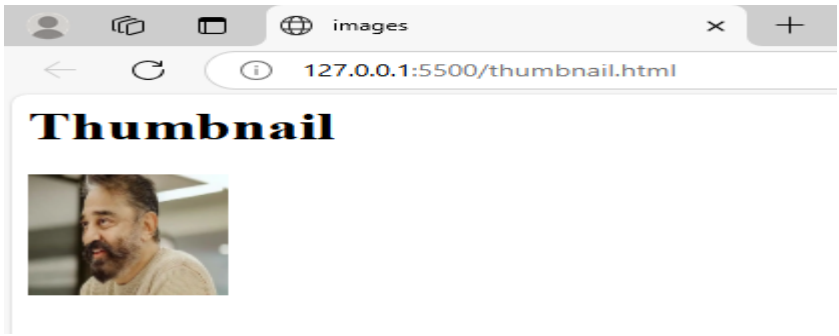
This approach is widely utilized in online galleries, e-commerce websites, and portfolio pages to create organized and visually appealing image collections. The program demonstrates the implementation of thumbnails linked to high-resolution images to build an interactive image gallery.

Programme:

```
<html>
<head>
   <title>images</title>
</head>
```

```
<body>
   <h1>Thumbnail</h1>
   <a href="https:// https://cdn.siasat.com/wp-content/uploads/2022/05/New-Project-1-780x470.jpg /" target="_blank">
       <img height=100px width=100px src="my_thumbnail.jpg" alt="kamal hasaan" width="150" height="150">
   </a>
</body>
</html>
```

OUTPUT:



Exercise-2. HTML Tables and Forms

A. Write a HTML program, to explain the working of tables. (usetags: <table>, <tr>, <th>, <td> and attributes: border, row span, col span)

Introduction

Tables in HTML are used to organize and display data in a structured format. The tag defines the table structure, while represents table rows. Each row consists of table headers () and table data (), which are used to label and display content respectively.

To enhance table formatting, attributes such as are used to define table outlines, while and allow merging of cells across multiple rows and columns, ensuring better alignment and readability.

Tables are widely utilized for presenting data in various web applications, including reports, schedules, and comparison charts. This program demonstrates the use of table-related tags and attributes to create well-organized tabular data.

PROGRAMME:

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML Table Example</title>
</head>
<body>

  <h1>Working with Tables in HTML</h1>

  <table border="1">
    <tr>
      <th rowspan="2">Name</th>
      <th colspan="2">Marks</th>
      <th rowspan="2">Total</th>
    </tr>
    <tr>
      <th>Math</th>
      <th>Science</th>
    </tr>
    <tr>
      <td>Alice</td>
      <td>85</td>
      <td>90</td>
      <td>175</td>
    </tr>
    <tr>
      <td>Bob</td>
      <td>78</td>
      <td>88</td>
      <td>166</td>
    </tr>
    <tr>
```

```
    <td>Charlie</td>
    <td>92</td>
    <td>81</td>
    <td>173</td>
   </tr>
  </table>

</body>
</html>
```

OUTPUT:

Working with Tables in HTML

Name	Marks		Total
	Math	Science	
Alice	85	90	175
Bob	78	88	166
Charlie	92	81	173

B.**Write a HTML program, to explain the working of tables by preparing a timetable. (Note: Use <caption> tag to set the caption to the table & also use cellspacing, cellpadding, border, rowspan, colspan etc.).**

Introduction

HTML tables are commonly used to present structured information in an organized manner. A timetable is an excellent example where tables help display schedules clearly. The tag defines the table structure, while represents rows, and or defines individual cells for headings and data respectively.

To enhance table formatting, attributes such as are used to create visible outlines, while and help adjust spacing between cells for better readability. Additionally, the tag provides a descriptive label for the table, and allow merging of cells across multiple rows or columns to properly format schedules.

This program illustrates how to structure a timetable using HTML tables, demonstrating the effective use of various attributes to create a well-organized and visually appealing schedule layout.

Programme:

```
<!DOCTYPE html>
```

```
<html>
<head>
 <title>Class Timetable</title>
</head>
<body>

 <h1>Weekly Class Timetable</h1>

 <table border="1" cellspacing="5" cellpadding="10">
  <caption><strong>Class Timetable - B.Tech CSE (2nd Year)</strong></caption>

  <tr>
   <th>Day / Time</th>
   <th>9:00 - 10:00</th>
   <th>10:00 - 11:00</th>
   <th>11:00 - 12:00</th>
   <th>12:00 - 1:00</th>
   <th>1:00 - 2:00</th>
   <th>2:00 - 3:00</th>
   <th>3:00 - 4:00</th>
  </tr>

  <tr>
   <th>Monday</th>
   <td>Math</td>
   <td>OS</td>
   <td>DBMS</td>
   <td>DS</td>
   <td rowspan="5">Lunch</td>
   <td colspan="2">Lab</td>
  </tr>
```

```
<tr>
 <th>Tuesday</th>
 <td>DBMS</td>
 <td>Math</td>
 <td>DS</td>
 <td>OS</td>
 <td>English</td>
 <td>Sports</td>
</tr>

<tr>
 <th>Wednesday</th>
 <td colspan="2">Lab</td>
 <td>DS</td>
 <td>Math</td>
 <td>OS</td>
 <td>Seminar</td>
</tr>

<tr>
 <th>Thursday</th>
 <td>English</td>
 <td>Math</td>
 <td>DBMS</td>
 <td>OS</td>
 <td colspan="2">Project Work</td>
</tr>

<tr>
 <th>Friday</th>
```

```
    <td>Seminar</td>
    <td>DS</td>
    <td>OS</td>
    <td>DBMS</td>
    <td>Math</td>
    <td>English</td>
  </tr>

 </table>

</body>
</html>
```

OUTPUT:

Weekly Class Timetable

Class Timetable - B.Tech AIML (2nd Year)

Day / Time	9:00 - 10:00	10:00 - 11:00	11:00 - 12:00	12:00 - 1:00	1:00 - 2:00	2:00 - 3:00	3:00 - 4:00
Monday	Math	OS	DBMS	DS	Lunch	Lab	
Tuesday	DBMS	Math	DS	OS		English	Sports
Wednesday	Lab		DS	Math		OS	Seminar
Thursday	English	Math	DBMS	OS		Project Work	
Friday	Seminar	DS	OS	DBMS		Math	English

C.Write a HTML program, to explain the working of forms by designing Registration form.(Note: Include text field, password field, number field, date of birth field, check boxes, radio buttons, list boxes using <select>&<option> tags, <text area> and two buttons ie: submit and reset. Use tables to provide a better view).

Introduction

Forms in HTML are essential for collecting user input and facilitating interactions on websites. They provide a structured way to gather data such as personal information, preferences, and selections. The tag is used to create a form, while various input elements such as text fields, password fields, number fields, date pickers, checkboxes, radio buttons, dropdown lists (and), and text areas are incorporated to enhance functionality.

To improve the visual arrangement and readability of the form, tables (, ,) can be used for structuring input fields systematically. Additionally, buttons like "Submit" and "Reset" allow users to either send the provided data or clear the inputs to re-enter information.

This program demonstrates the use of HTML forms by designing a registration form that includes multiple input types, ensuring a user-friendly and well-organized structure for data collection.

PROGRAMME:

```
<!DOCTYPE html>
<html>
<head>
 <title>Registration Form</title>
</head>
<body>

 <h1>User Registration Form</h1>

 <form action="#" method="post">
  <table border="1" cellpadding="10" cellspacing="0">
   <caption><strong>Registration Form</strong></caption>

   <tr>
    <td><label for="fullname">Full Name:</label></td>
```

```
  <td><input type="text" id="fullname" name="fullname" required></td>
</tr>

<tr>
  <td><label for="password">Password:</label></td>
  <td><input type="password" id="password" name="password" required></td>
</tr>

<tr>
  <td><label for="age">Age:</label></td>
  <td><input type="number" id="age" name="age" min="1" max="100"></td>
</tr>

<tr>
  <td><label for="dob">Date of Birth:</label></td>
  <td><input type="date" id="dob" name="dob"></td>
</tr>

<tr>
  <td>Gender:</td>
  <td>
    <input type="radio" id="male" name="gender" value="Male">
    <label for="male">Male</label>
    <input type="radio" id="female" name="gender" value="Female">
    <label for="female">Female</label>
    <input type="radio" id="other" name="gender" value="Other">
    <label for="other">Other</label>
  </td>
</tr>
```

```
<tr>
 <td>Languages Known:</td>
 <td>
  <input type="checkbox" id="english" name="lang" value="English">
  <label for="english">English</label>
  <input type="checkbox" id="hindi" name="lang" value="Hindi">
  <label for="hindi">Hindi</label>
  <input type="checkbox" id="telugu" name="lang" value="Telugu">
  <label for="telugu">Telugu</label>
 </td>
</tr>

<tr>
 <td><label for="course">Select Course:</label></td>
 <td>
  <select id="course" name="course">
   <option value="CSE">Computer Science</option>
   <option value="ECE">Electronics</option>
   <option value="EEE">Electrical</option>
   <option value="MECH">Mechanical</option>
   <option value="CIVIL">Civil</option>
  </select>
 </td>
</tr>

<tr>
 <td><label for="address">Address:</label></td>
 <td><textarea id="address" name="address" rows="4" cols="30"></textarea></td>
```

```
    </tr>

    <tr>
      <td colspan="2" align="center">
        <input type="submit" value="Submit">
        <input type="reset" value="Reset">
      </td>
    </tr>

  </table>
 </form>

</body>
</html>
```

OUTPUT:

User Registration Form

Registration Form

Full Name:	
Password:	
Age:	
Date of Birth:	dd/mm/yyyy
Gender:	Male Female Other
Languages Known:	English Hindi Telugu
Select Course:	Computer Science
Address:	
Submit Reset	

D. Write a HTML program, that makes use of <article>, <aside>, <figure>, <figcaption>, <footer>, <header>, <main>, <nav>, <section>, , tags.

Introduction: Semantic and Structural Tags in HTML

HTML (HyperText Markup Language) is the standard language used to create and structure content on the web. As websites have evolved, so has the need for **clean**, **readable**, and **accessible** code. To support this, **HTML5 introduced semantic tags** that clearly describe the purpose of web elements, improving both **SEO (Search Engine Optimization)** and **accessibility**.

In this program, we explore the use of important **semantic** and **structural** tags such as:

- `<header>` – Represents introductory content or a navigational heading for a section or page.
- `<nav>` – Denotes a navigation menu for links.
- `<main>` – Contains the central content unique to the page.
- `<section>` – Groups content with a common theme.
- `<article>` – Defines independent, self-contained content like blog posts or news articles.
- `<aside>` – Holds side content, like advertisements or additional info.
- `<figure>` – Used to enclose media content such as images.
- `<figcaption>` – Provides a caption or description for the media inside `<figure>`.
- `<footer>` – Contains the footer or bottom section of a document or section.
- `<div>` – A generic container used to group elements for styling or scripting purposes.
- `<span>` – An inline container used to style parts of text or group small chunks of content.

These tags enhance the **semantic meaning** of the webpage, make it easier for browsers and screen readers to understand the structure, and support better styling with CSS and functionality with JavaScript.

This program demonstrates how these elements work together to form a well-structured and modern HTML5 page.

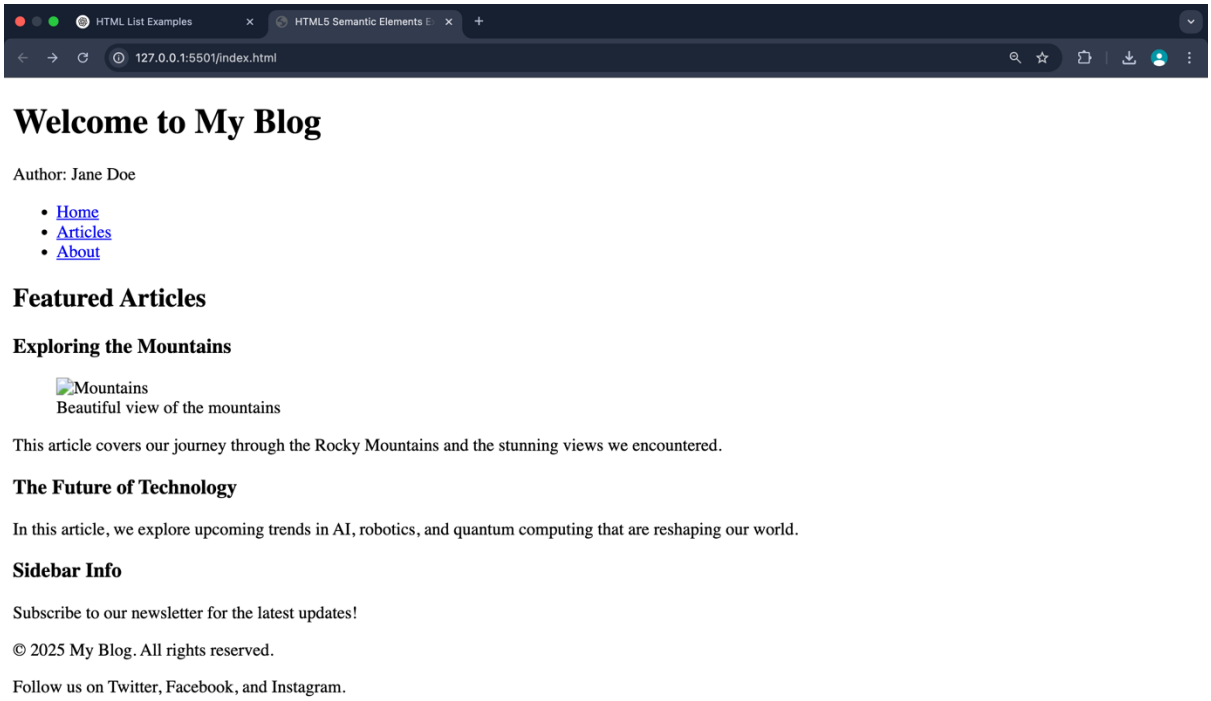
PROGRAMME:

```
<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <title>HTML5 Semantic Elements Example</title>

</head>
```

```
<body>

 <header>
  <h1>Welcome to My Blog</h1>
  <p><span>Author:</span> Jane Doe</p>
 </header>

 <nav>
  <ul>
   <li><a href="#home">Home</a></li>
   <li><a href="#articles">Articles</a></li>
   <li><a href="#about">About</a></li>
  </ul>
 </nav>

 <main>
  <section id="articles">
   <h2>Featured Articles</h2>

   <article>
    <h3>Exploring the Mountains</h3>
    <figure>
     <img src="mountains.jpg" alt="Mountains" width="300" height="200">
     <figcaption>Beautiful view of the mountains</figcaption>
    </figure>
    <p>This article covers our journey through the Rocky Mountains and the stunning views we encountered.</p>
   </article>

   <article>
```

```
      <h3>The Future of Technology</h3>
      <p>In this article, we explore upcoming trends in AI, robotics, and quantum computing that are reshaping our world.</p>
    </article>
  </section>

  <aside>
    <h3>Sidebar Info</h3>
    <p>Subscribe to our newsletter for the latest updates!</p>
  </aside>
 </main>

 <footer>
  <div>
    <p>&copy; 2025 My Blog. All rights reserved.</p>
    <p>Follow us on <span>Twitter</span>, <span>Facebook</span>, and <span>Instagram</span>.</p>
  </div>
 </footer>

</body>
</html>
```

OUTPUT:



E. Write a HTML program, to embed audio and video into HTML webpage.

Introduction: Embedding Audio and Video in HTML

With the evolution of the web, multimedia elements like **audio** and **video** have become essential for creating engaging and interactive webpages. HTML5 has made this integration simpler by introducing the `<audio>` and `<video>` tags, allowing developers to embed media directly into web pages **without the need for external plugins** like Flash.

- The `<audio>` tag is used to **embed audio content** such as music, podcasts, or sound effects.
- The `<video>` tag is used to **embed video files** directly onto a webpage, like tutorials, promos, or educational clips.

These tags support several attributes to enhance user experience:

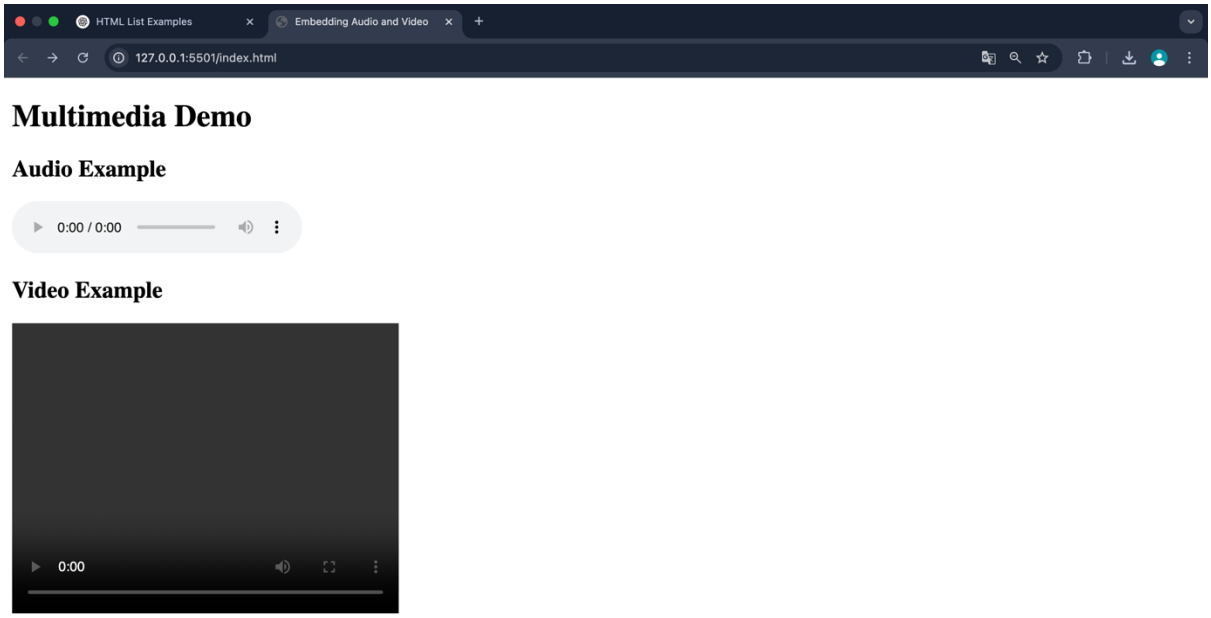
- `controls` – Displays playback controls (play, pause, volume).
- `autoplay` – Automatically starts playing the media when the page loads.
- `loop` – Repeats the media after it ends.
- `muted` – Starts the media without sound.
- `preload` – Suggests how the browser should load the media (none, metadata, auto).
- `src` – Specifies the path to the media file.
- `<source>` – Allows specifying multiple file formats for better browser support.

This program demonstrates how to embed both audio and video clips using HTML5 tags, creating a media-rich and user-friendly webpage.

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Audio & video</title>
</head>
<body>
   <h1>Audio Tag</h1>
   <audio controls autoplay >
      <source src="Hi Nanna Song Bgm Ringtone Download – Nani - MobCup.Com.Co.mp3"
type="audio/mpeg">
      your browser doesn't support audio tag
   </audio>

   <h1>The video element</h1>

<video width="320" height="240" controls>
  <source src="170204-842720416_small.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>

</body>
</html>
```

OUTPUT:


HTML List Examples
Embedding Audio and Video
127.0.0.1:5501/index.html
Multimedia Demo
Audio Example
0:00 / 0:00
Video Example
0:00

Exercise-3.Cascading Style Sheets, Selector forms

Write a program to apply different types of selector forms

i **Simple selector (element, id, class, group, universal)**
ii **Combinator selector (descendant, child, adjacent sibling, general sibling)**
iii **Pseudo-class selector**
iv **Pseudo-element selector**
v **Attribute selector**

Introduction: Exploring CSS Selectors in Web Design

Cascading Style Sheets (CSS) are essential for designing visually appealing, structured, and interactive web pages. At the heart of CSS lies the concept of **selectors**, which allow developers to target specific HTML elements and apply styles effectively. Mastering selectors is key to building clean, efficient, and dynamic web designs.

In this program, we explore different **forms of CSS selectors** used to select and style HTML elements with precision:

---

1. Simple Selectors

These are the most commonly used and include:

- **Element Selector** – Targets HTML tags like `p`, `h1`, `div`.
- **ID Selector** (`#id`) – Styles an element with a unique ID.
- **Class Selector** (`.class`) – Targets multiple elements sharing the same class.
- **Group Selector** (`h1, p, div`) – Applies the same styles to a group of elements.
- **Universal Selector** (`*`) – Targets all elements on the page.

---

2. Combinator Selectors

Used to define relationships between elements:

- **Descendant (`A B`)** – Targets `B` inside `A` at any level.
- **Child (`A > B`)** – Targets `B` only if it's a direct child of `A`.
- **Adjacent Sibling (`A + B`)** – Targets `B` if it comes immediately after `A`.
- **General Sibling (`A ~ B`)** – Targets all `B` siblings that follow `A`.

---

3. Pseudo-Class Selectors

These target elements in a particular state:

- Examples: `:hover`, `:focus`, `:first-child`, `:last-child`, `:nth-child(n)`, etc.

---

4. Pseudo-Element Selectors

These target parts of an element's content:

- Examples: `::before`, `::after`, `::first-letter`, `::first-line`, etc.

---

5. Attribute Selectors

These select elements based on the presence or value of attributes:

- Examples: `[type="text"]`, `[href^="https"]`, `[alt*="logo"]`, etc.

This program demonstrates the application of all these selector types to showcase how CSS can be used to precisely style various parts of a webpage, enhancing both the **structure and user interaction** of the design.

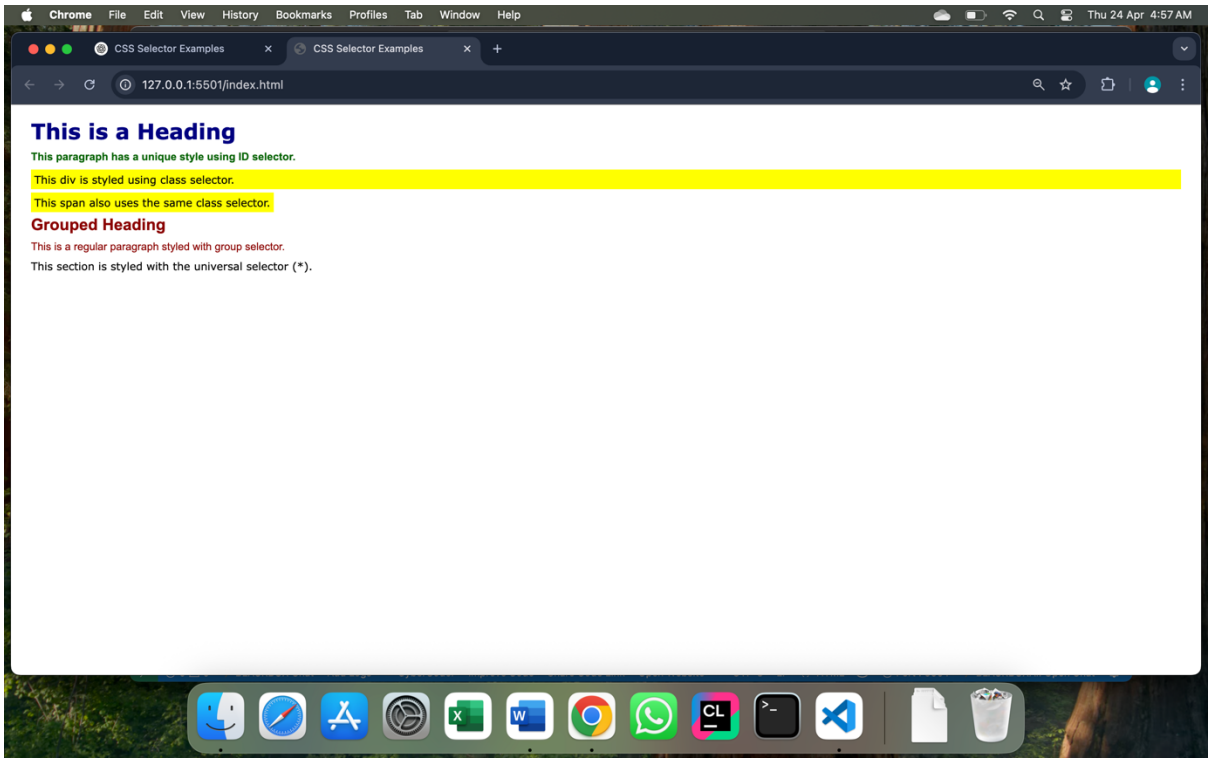
PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>CSS Selector Examples</title>
 <style>
  /* 1. Element Selector */
  h1 {
   color: navy;
   font-size: 2em;
  }

  /* 2. ID Selector */
  #special-para {
   color: darkgreen;
```

```
    font-weight: bold;
   }

   /* 3. Class Selector */
   .highlight {
    background-color: yellow;
    padding: 5px;
   }

   /* 4. Group Selector */
   h2, p {
    color: darkred;
    font-family: Arial, sans-serif;
   }

   /* 5. Universal Selector */
   * {
    margin: 10px;
    font-family: Verdana, sans-serif;
   }
  </style>
</head>
<body>

  <!-- Element Selector -->
  <h1>This is a Heading</h1>

  <!-- ID Selector -->
  <p id="special-para">This paragraph has a unique style using ID selector.</p>
```

```
 <!-- Class Selector -->
 <div class="highlight">This div is styled using class selector.</div>
 <span class="highlight">This span also uses the same class selector.</span>
<!-- Group Selector -->
 <h2>Grouped Heading</h2>
 <p>This is a regular paragraph styled with group selector.</p>
 <!-- Universal Selector -->
 <section>This section is styled with the universal selector (*).</section>
</body>
</html>
```

OUTPUT:



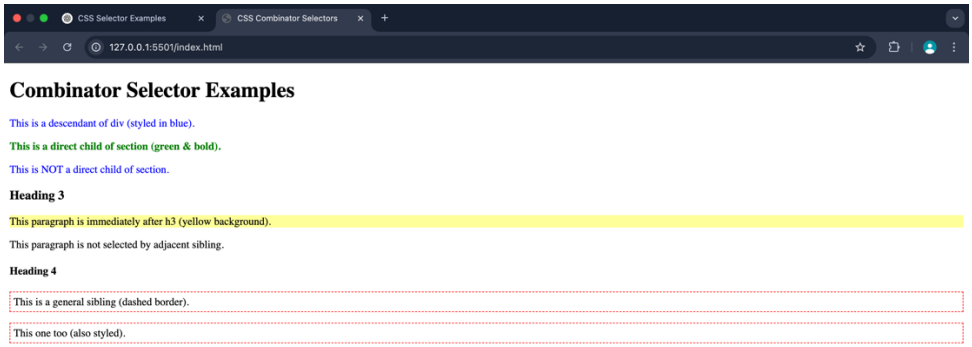
ii. Combinator selector (descendant, child, adjacent sibling, general sibling)

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>CSS Combinator Selectors</title>
 <style>
  /* 1. Descendant Selector: any <p> inside <div> (nested at any level) */
  div p {
   color: blue;
  }
  /* 2. Child Selector: <p> that is a direct child of <section> */
  section > p {
   color: green;
   font-weight: bold;
  }
  /* 3. Adjacent Sibling Selector: <p> immediately after an <h3> */
  h3 + p {
   background-color: #ffff99;
  }
  /* 4. General Sibling Selector: all <p> after an <h4> that share the same parent */
  h4 ~ p {
   border: 1px dashed red;
   padding: 5px;
  }
 </style>
</head>
<body>
```

```
 <h1>Combinator Selector Examples</h1>
 <!-- Descendant Selector -->
 <div>
  <article>
   <p>This is a descendant of div (styled in blue).</p>
  </article>
 </div>
 <!-- Child Selector -->
 <section>
  <p>This is a direct child of section (green & bold).</p>
  <div>
   <p>This is NOT a direct child of section.</p>
  </div>
 </section>
 <!-- Adjacent Sibling Selector -->
 <h3>Heading 3</h3>
 <p>This paragraph is immediately after h3 (yellow background).</p>
 <p>This paragraph is not selected by adjacent sibling.</p>
 <!-- General Sibling Selector -->
 <h4>Heading 4</h4>
 <p>This is a general sibling (dashed border).</p>
 <p>This one too (also styled).</p>
</body>
</html>
```

OUTPUT:



iii. Pseudo-class selector
iv. Pseudo-element selector

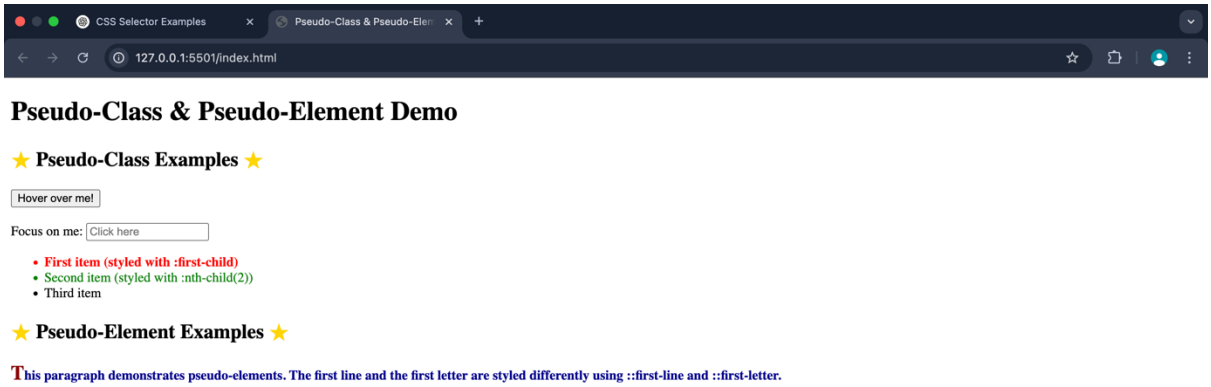
PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Pseudo-Class & Pseudo-Element Selectors</title>
 <style>
  /* :hover - Change color on mouse hover */
  button:hover {
   background-color: #4CAF50;
   color: white;
   cursor: pointer;
  }
  ul li:first-child {
   color: red;
   font-weight: bold;
  }
  ul li:nth-child(2) {
```

```
    color: green;
   }
   input:focus {
    border: 2px solid blue;
    outline: none;
   }
   /* ===== PSEUDO-ELEMENT SELECTORS ===== */
   /* ::before - Add content before element */
   h2::before {
    content: "★ ";
    color: gold;
   }
   /* ::after - Add content after element */
   h2::after {
    content: " ★";
    color: gold;
   }
   /* ::first-line - Style first line of paragraph */
   p::first-line {
    font-weight: bold;
    color: darkblue;
   }
   /* ::first-letter - Style first letter of paragraph */
   p::first-letter {
    font-size: 150%;
    color: darkred;
   }
  </style>
</head>
<body>
  <h1>Pseudo-Class & Pseudo-Element Demo</h1>
  <!-- Pseudo-Class -->
  <h2>Pseudo-Class Examples</h2>
```

```
<button>Hover over me!</button>
<br><br>
<label for="inputBox">Focus on me:</label>
<input type="text" id="inputBox" placeholder="Click here">
<ul>
  <li>First item (styled with :first-child)</li>
  <li>Second item (styled with :nth-child(2))</li>
  <li>Third item</li>
</ul>
<!-- Pseudo-Element -->
<h2>Pseudo-Element Examples</h2>
<p>This paragraph demonstrates pseudo-elements. The first line and the first letter are styled differently using ::first-line and ::first-letter.</p>
</body>
</html>
```

OUTPUT:



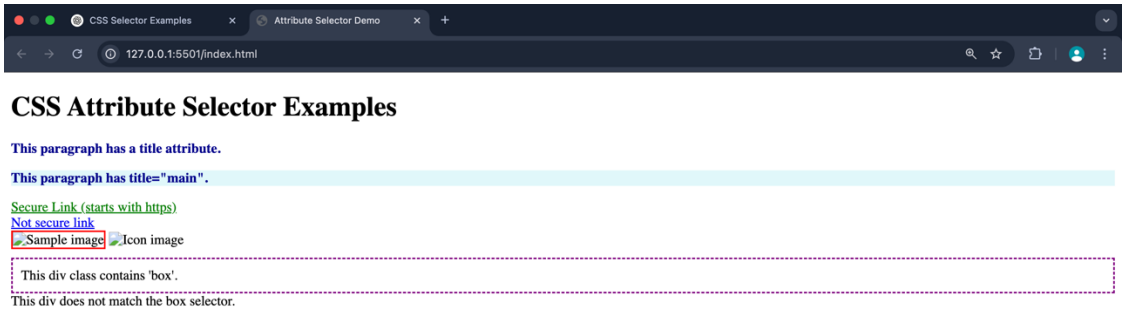
v.Attribute selector

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Attribute Selector Demo</title>
 <style>
  /* 1. Select all elements with a title attribute */
  [title] {
   color: darkblue;
   font-weight: bold;
  }
  /* 2. Select elements with title exactly "main" */
  [title="main"] {
   background-color: #e0f7fa;
  }
  /* 3. Select elements where href starts with "https" */
  a[href^="https"] {
   color: green;
  }
  /* 4. Select elements where src ends with ".jpg" */
  img[src$=".jpg"] {
   border: 2px solid red;
  }
  /* 5. Select elements where class contains "box" */
  div[class*="box"] {
   border: 2px dashed purple;
   padding: 10px;
   margin-top: 10px;
  }
 </style>
```

```
</head>
<body>
 <h1>CSS Attribute Selector Examples</h1>
 <!-- Attribute present -->
 <p title="example">This paragraph has a title attribute.</p>
 <!-- Attribute equals -->
 <p title="main">This paragraph has title="main".</p>
 <!-- Starts with -->
 <a href="https://example.com">Secure Link (starts with https)</a><br>
 <a href="http://example.com">Not secure link</a><br>
 <!-- Ends with -->
 <img src="photo.jpg" alt="Sample image" width="150">
 <img src="icon.png" alt="Icon image" width="150"><br>
 <!-- Contains -->
 <div class="highlight-box">This div class contains 'box'.</div>
 <div class="container">This div does not match the box selector.</div>
</body>
</html>
```

OUTPUT:



Exercise-4.Types of CSS, CSS with Color, Background, Font, Text and CSS Box Model

Write a program to apply different types (or levels of styles or style specification formats) - inline, internal, external styles to HTML elements. (Identify selector, property and value).

Introduction: Understanding Types of CSS and Styling Techniques in Web Development

Cascading Style Sheets (CSS) form the backbone of web design by providing powerful tools to enhance the appearance of HTML elements. One of the first steps toward mastering CSS is understanding the **three main types of CSS** application techniques and learn how to style elements using various properties such as **color**, **background**, **font**, **text**, and the **box model**.

📘 Types of CSS (Levels of Style Specification)

1. **Inline CSS**
   - Styles are applied directly within HTML elements using the `style` attribute.
   - Best for quick styling, but not recommended for larger projects due to poor readability and maintenance.
2. **Internal CSS**
   - Defined within the `<style>` tag inside the HTML document's `<head>`.
   - Useful when styling a single HTML file.
3. **External CSS**
   - Written in a separate `.css` file and linked using `<link>` in the `<head>`.
   - Ideal for styling multiple web pages consistently.

Each method includes a **selector** (which targets an element), a **property** (what you want to change), and a **value** (the new appearance or behavior).

🌈 CSS Styling Properties in Focus

1. **Color Properties**
   - `color`: Sets the text color.
   - `background-color`: Sets the background color of an element.
2. **Background Styling**
   - `background-image`, `background-repeat`, `background-position`, `background-size`.
3. **Font Properties**
   - `font-family`, `font-size`, `font-style`, `font-weight`.
4. **Text Properties**
   - `text-align`, `text-decoration`, `line-height`, `letter-spacing`.
5. **Box Model Concepts**
   - The **Box Model** includes `margin`, `border`, `padding`, and `width/height`.

- Understanding how these parts interact is essential for spacing and layout control.

👨‍💻 What You'll Learn Through This Program

- How to apply **each CSS type** to various HTML elements.
- How to **identify and use selectors** (element, class, ID).
- How to apply **essential styling properties** to change the layout and design of content.
- How the **box model** affects spacing and how to manipulate it for perfect alignment.

This foundational knowledge empowers developers to create **aesthetic**, **responsive**, and **organized** web pages with greater control and flexibility.

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSS Style Types</title>
  <!-- External CSS -->
  <link rel="stylesheet" href="style.css"> <!-- Simulated external file -->
  <!-- Internal CSS -->
  <style>
    /* Internal Style */
    /* Selector: h2 | Property: color | Value: green */
    h2 {
      color: green;
    }

    /* Selector: .internal-box | Property: background-color | Value: lightblue */
    .internal-box {
      background-color: lightblue;
      padding: 10px;
    }
  </style>
</head>
<body>

  <h1 style="color: red;"> <!-- Inline Style -->
    <!-- Selector: h1 | Property: color | Value: red -->
    This heading uses an inline style (red color)
  </h1>

  <h2>This heading uses internal style (green color)</h2>

  <div class="internal-box">
    This div uses internal style via class `.internal-box`
  </div>

  <div class="external-box">
    This div will be styled using external CSS
  </div>
```

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

Style.css

```
/* External CSS */
/* Selector: .external-box | Property: background-color | Value: lightgray */
.external-box {
  background-color: lightgray;
  padding: 10px;
  border: 2px solid black;
}
```

OUTPUT:

B. Write a program to demonstrate the various ways you can reference a color in CSS.

Introduction: Demonstrating Various Ways to Reference Colors in CSS

Color plays a significant role in web design by enhancing the visual appeal, establishing brand identity, and improving user experience. In Cascading Style Sheets (CSS), there are multiple ways to define and apply colors to HTML elements. Each method offers different levels of control and readability, allowing web developers to select the most appropriate approach based on their design requirements.

This program is designed to demonstrate the various methods available for specifying colors in CSS. The objective is to help learners understand how different color formats work and how they can be implemented effectively within a webpage.

Common Methods to Reference Colors in CSS

1. **Named Colors**
   CSS provides a predefined set of color names such as `red`, `blue`, `green`, `black`, and `white`. These are easy to use and remember.
2. **Hexadecimal Notation**
   Hex codes are widely used to represent colors. They consist of six digits and are prefixed with a hash symbol (#), for example, `#FF5733`.
3. **RGB (Red, Green, Blue)**
   This method defines colors using the RGB color model, where each component can have a value between 0 and 255. An example is `rgb(255, 99, 71)`.
4. **RGBA (RGB with Alpha Transparency)**
   RGBA is an extension of the RGB model that includes an alpha parameter to control transparency. Example: `rgba(0, 128, 0, 0.5)`.
5. **HSL (Hue, Saturation, Lightness)**
   HSL is a more intuitive way to define colors based on hue (measured in degrees), saturation, and lightness (both in percentage). For example: `hsl(240, 100%, 50%)`.
6. **HSLA (HSL with Alpha Transparency)**
   HSLA adds an alpha channel to the HSL model to specify the level of transparency. Example: `hsla(240, 100%, 50%, 0.3)`.

PROGRAMME:

```
<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">
```

```
<title>Color Reference Examples</title>
<style>
 /* Named Color */
 .named-color {
   background-color: red;
   color: white;
   padding: 10px;
 }
 /* Hexadecimal */
 .hex-color {
   background-color: #00FF00; /* green */
   color: black;
   padding: 10px;
 }
 /* RGB */
 .rgb-color {
   background-color: rgb(0, 0, 255); /* blue */
   color: white;
   padding: 10px;
 }
 /* RGBA (with opacity) */
 .rgba-color {
   background-color: rgba(255, 0, 0, 0.5); /* semi-transparent red */
   color: black;
   padding: 10px;
 }
 /* HSL */
 .hsl-color {
   background-color: hsl(120, 100%, 50%); /* green */
   color: white;
   padding: 10px;
 }
 /* HSLA */
```

```
   .hsla-color {
     background-color: hsla(240, 100%, 50%, 0.5); /* semi-transparent blue */
     color: white;
     padding: 10px;
    }
  </style>
</head>
<body>
  <h1>CSS Color Reference Formats</h1>
  <div class="named-color">This uses a <strong>named color</strong>: red</div>
  <div class="hex-color">This uses a <strong>hex color</strong>: #00FF00</div>
  <div class="rgb-color">This uses an <strong>RGB color</strong>: rgb(0, 0, 255)</div>
  <div class="rgba-color">This uses an <strong>RGBA color</strong>: rgba(255, 0, 0, 0.5)</div>
  <div class="hsl-color">This uses an <strong>HSL color</strong>: hsl(120, 100%, 50%)</div>
  <div class="hsla-color">This uses an <strong>HSLA color</strong>: hsla(240, 100%, 50%, 0.5)</div>


</body>
</html>
```

OUTPUT:

C. Write a CSS rule that places a background image halfway down the page, tilting it horizontally. The image should remain in place when the user scrolls up or down.

Introduction: Background Image Placement and Scrolling Behavior in CSS

In web development, the use of background images enhances the visual appeal of a webpage. CSS provides powerful properties to control the position, repetition, attachment, and transformation of background images. This exercise demonstrates how to position a background image halfway down the page, apply a horizontal transformation (tilt), and keep the image fixed during scrolling, thereby improving the design aesthetics and user experience.

The following CSS rule utilizes key properties like `background-position`, `background-attachment`, and `transform` to accomplish this behavior.

CSS Rule

```
css
Copy code
body {
  background-image: url('your-image.jpg');
  background-repeat: no-repeat;
  background-position: center 50%;
  background-attachment: fixed;
  transform: scaleX(-1);
}
```

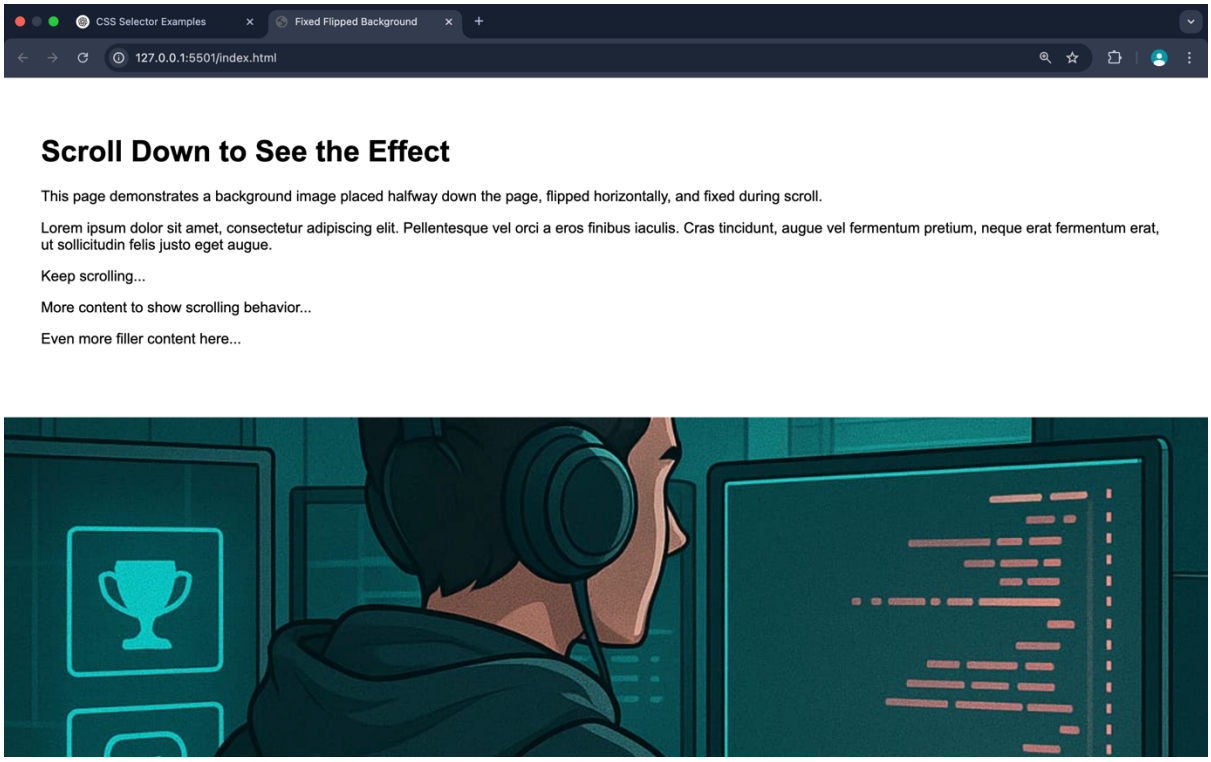
Explanation of Properties Used

- **`background-image`**: Sets the image to be used as the background.
- **`background-repeat: no-repeat`**: Ensures the image does not repeat across the page.
- **`background-position: center 50%`**: Positions the image horizontally at the center and vertically at 50% of the viewport height (halfway down).
- **`background-attachment: fixed`**: Makes the background image stay in place even when the user scrolls.
- **`transform: scaleX(-1)`**: Flips the image horizontally, creating a mirror (tilted) effect.

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Fixed Flipped Background</title>
  <style>
    /* Basic page styling */
    body {
      margin: 0;
      font-family: Arial, sans-serif;
      min-height: 200vh; /* To allow scrolling */
      position: relative;
    }

    /* Pseudo-element for background image */
    body::before {
      content: '';
      position: fixed;
      top: 50%; /* Halfway down the page */
      left: 0;
      width: 100%;
      height: 100%;
      background-image: url('your-image.jpg'); /* 🔁 Replace with your image path */
      background-size: cover;
      background-repeat: no-repeat;
      background-position: center;
      transform: scaleX(-1); /* Horizontal flip */
      z-index: -1; /* Behind content */
      pointer-events: none; /* So it doesn't block clicks */
    }

    /* Content */
    .content {
      padding: 40px;
    }
  </style>
</head>
<body>
  <div class="content">
    <h1>Scroll Down to See the Effect</h1>
    <p>This page demonstrates a background image placed halfway down the page, flipped horizontally, and fixed during scroll.</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque vel orci a eros finibus iaculis. Cras tincidunt, augue vel fermentum pretium, neque erat fermentum erat, ut sollicitudin felis justo eget augue.</p>
    <p>Keep scrolling...</p>
    <p>More content to show scrolling behavior...</p>
    <p>Even more filler content here...</p>
  </div>
</body>
</html>
```

OUTPUT:



D. Write a program using the following terms related to CSS font and text:
font-size
font-weight
font-style
text-decoration
text-transformation
text-alignment

Introduction

In web development, the appearance of text plays a crucial role in user experience and readability. CSS (Cascading Style Sheets) provides a wide array of properties that allow developers to control how text is displayed. By adjusting font properties such as **font-size**, **font-weight**, **font-style**, and **text-decoration**, you can customize the visual presentation of text to suit your design needs. Additionally, **text-transformation** and **text-align** offer more control over text case and alignment within its container.

The following program demonstrates the usage of various CSS properties related to font and text, including:

font-size: Sets the size of the text.
font-weight: Controls the thickness of the text (e.g., bold, normal).
font-style: Defines the style of the font, such as italic or normal.
text-decoration: Adds special effects like underline or strikethrough.
text-transform: Alters the text case (e.g., converting to uppercase).
text-align: Adjusts the alignment of text (left, right, or center).

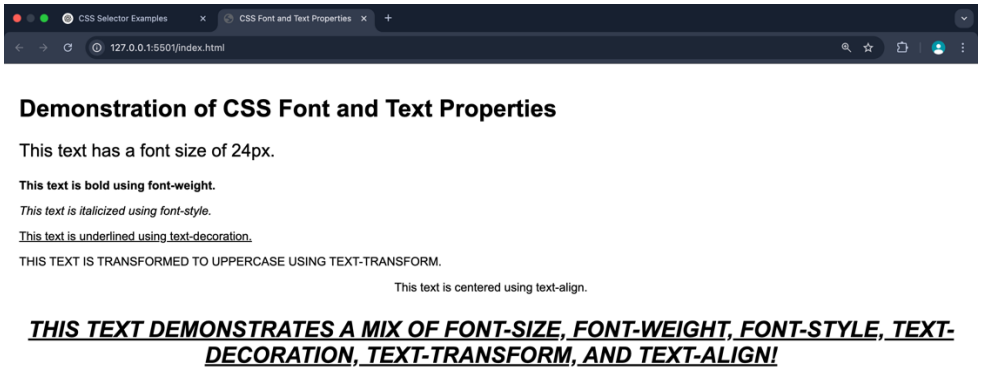
PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>CSS Font and Text Properties</title>
 <style>
  /* General body styling */
  body {
   font-family: Arial, sans-serif;
   margin: 0;
   padding: 20px;
  }
  /* Font-size: 24px */
  .font-size-example {
   font-size: 24px; /* Font size set to 24px */
  }
  /* Font-weight: bold */
  .font-weight-example {
   font-weight: bold; /* Makes the text bold */
  }
  /* Font-style: italic */
  .font-style-example {
   font-style: italic; /* Italicizes the text */
  }
  /* Text-decoration: underline */
  .text-decoration-example {
   text-decoration: underline; /* Underlines the text */
  }
  /* Text-transform: uppercase */
  .text-transform-example {
   text-transform: uppercase; /* Transforms text to uppercase */
```

```
    }
    /* Text-align: center */
    .text-align-example {
      text-align: center; /* Centers the text */
    }
    /* Mixed Example */
    .mixed-example {
      font-size: 30px;
      font-weight: bold;
      font-style: italic;
      text-decoration: underline;
      text-transform: uppercase;
      text-align: center;
    }
  </style>
</head>
<body>
  <h1>Demonstration of CSS Font and Text Properties</h1>
  <div class="font-size-example">
    <p>This text has a font size of 24px.</p>
  </div>
  <div class="font-weight-example">
    <p>This text is bold using font-weight.</p>
  </div>
  <div class="font-style-example">
    <p>This text is italicized using font-style.</p>
  </div>
  <div class="text-decoration-example">
    <p>This text is underlined using text-decoration.</p>
  </div>
  <div class="text-transform-example">
    <p>This text is transformed to uppercase using text-transform.</p>
  </div>
```

```
 <div class="text-align-example">
  <p>This text is centered using text-align.</p>
 </div>
 <div class="mixed-example">
  <p>This text demonstrates a mix of font-size, font-weight, font-style, text-decoration, text-transform, and text-align!</p>
 </div>
</body>
</html>
```

OUTPUT:



E. Write a program, to explain the importance of CSS Box model using

Content
Border
Margin
padding

Introduction

The **CSS Box Model** is a fundamental concept in web design, determining how elements are structured and spaced on a webpage. The model consists of several layers: **content**, **padding**, **border**, and **margin**. Each of these layers plays a key role in controlling the layout and positioning of elements on a webpage.

- **Content**: The actual content of the element, such as text or images.
- **Padding**: The space between the content and the border, used to create spacing inside the element.

- **Border**: The edge around the element, surrounding the padding.
- **Margin**: The space outside the border, separating the element from other elements.

Understanding how each layer of the box model works is crucial for positioning and designing elements correctly. Below is a simple program that demonstrates how these properties interact and affect an element's layout.

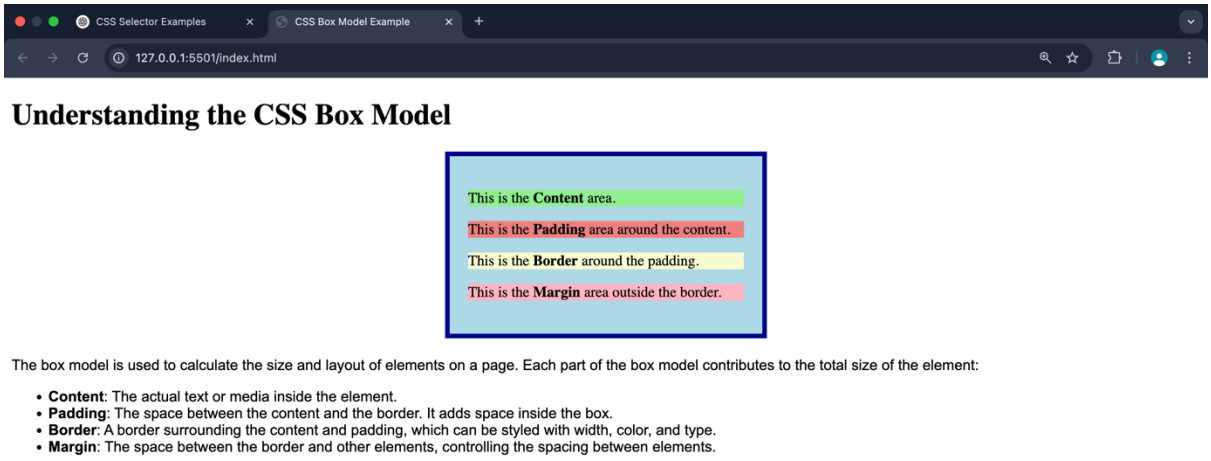
PROGRAMME:

```
<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <title>CSS Box Model Example</title>

 <style>

  /* Styling for box container */

  .box {

   width: 300px;

   background-color: lightblue;

   margin: 20px auto;

   padding: 20px;

   border: 5px solid darkblue;

  }

  /* Explanation text */

  .explanation {

   font-family: Arial, sans-serif;

   margin-top: 20px;

  }

  .content-box {

   background-color: lightgreen;

  }
```

```
    .padding-box {
      background-color: lightcoral;
    }
    .border-box {
      background-color: lightgoldenrodyellow;
    }
    .margin-box {
      background-color: lightpink;
    }
  </style>
</head>
<body>
  <h1>Understanding the CSS Box Model</h1>
  <div class="box">
    <div class="content-box">
      <p>This is the <strong>Content</strong> area.</p>
    </div>
    <div class="padding-box">
      <p>This is the <strong>Padding</strong> area around the content.</p>
    </div>
    <div class="border-box">
      <p>This is the <strong>Border</strong> around the padding.</p>
    </div>
    <div class="margin-box">
      <p>This is the <strong>Margin</strong> area outside the border.</p>
    </div>
  </div>
  <div class="explanation">
    <p>The box model is used to calculate the size and layout of elements on a page. Each part of the box model contributes to the total size of the element:</p>
```

```
    <ul>
      <li><strong>Content</strong>: The actual text or media inside the element.</li>
      <li><strong>Padding</strong>: The space between the content and the border. It adds space inside the box.</li>
      <li><strong>Border</strong>: A border surrounding the content and padding, which can be styled with width, color, and type.</li>
      <li><strong>Margin</strong>: The space between the border and other elements, controlling the spacing between elements.</li>
    </ul>
  </div>
</body>
</html>
```

OUTPUT:



Exercise-5. Applying JavaScript-internal and external, I/O, Type Conversion

Write a program to embed internal and external Java Script in a webpage.

Introduction

JavaScript is a versatile scripting language that adds interactivity and functionality to websites. It can be embedded in HTML in two primary ways:

1. **Internal JavaScript**: This involves writing JavaScript directly inside the HTML file using the `<script>` tag. The code can be placed in the `<head>` or `<body>` section of the HTML document.
2. **External JavaScript**: In this approach, JavaScript code is written in a separate file with a `.js` extension. This file is then linked to the HTML document using the `<script>` tag with the `src` attribute.

Both methods are useful depending on the context and size of the project. Internal JavaScript is commonly used for smaller scripts or when quick functionality is needed, while external JavaScript is preferred for larger applications to keep code modular, reusable, and easier to manage. By combining both internal and external JavaScript, developers can create dynamic and interactive web pages that enhance user experience.

4o mini

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JS Embedding</title>
  <script src="embed.js"></script>
  <style>
    body {
      margin: 0;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
    }
```

```
        h1 {
            text-align: center;
        }
        button {
            margin: 10px;
            background-color: mediumblue;
            color: aliceblue;
            text-align:center;
            border-color:lightpink
        }
    </style>
</head>
<body>
    <h1>JavaScript Embedding</h1>
    <div>
        <button onclick="internal()">Internal JS</button>
        <button onclick="external()">External JS</button>
    </div>
    <script>
        function internal() {
            alert(`This is from an internal message...`);
        }
    </script>
</body>
</html>
```

EMBED.JS

```
function external()
{
    alert(`This is from external JS`);
}
```

OUTPUT:

B. Write a program to explain the different ways for taking input and displaying output.

Introduction

In JavaScript, there are various ways to take input from users and display output. These methods can be categorized into:

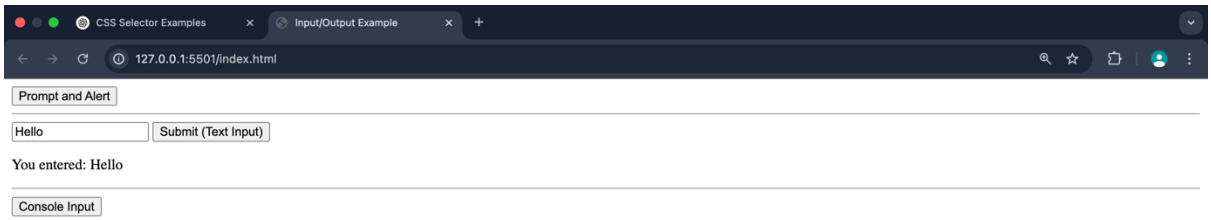
1. **Taking Input**:
   - **`prompt()`**: A built-in JavaScript function that allows the user to enter input via a dialog box.
   - **Form Elements**: You can use form elements like `<input>`, `<textarea>`, and `<select>` to take input in a more structured way.
   - **`console.read()`**: While not standard for browsers, it can be used in some environments for input (e.g., Node.js).
2. **Displaying Output**:
   - **`alert()`**: A simple built-in function that shows a pop-up dialog with a message.
   - **`console.log()`**: This is mainly used for debugging, displaying information in the browser's console.
   - **`document.write()`**: Allows you to directly write output to the HTML document.
   - **DOM Manipulation**: Using `innerHTML` or `textContent`, you can display dynamic content on the webpage.

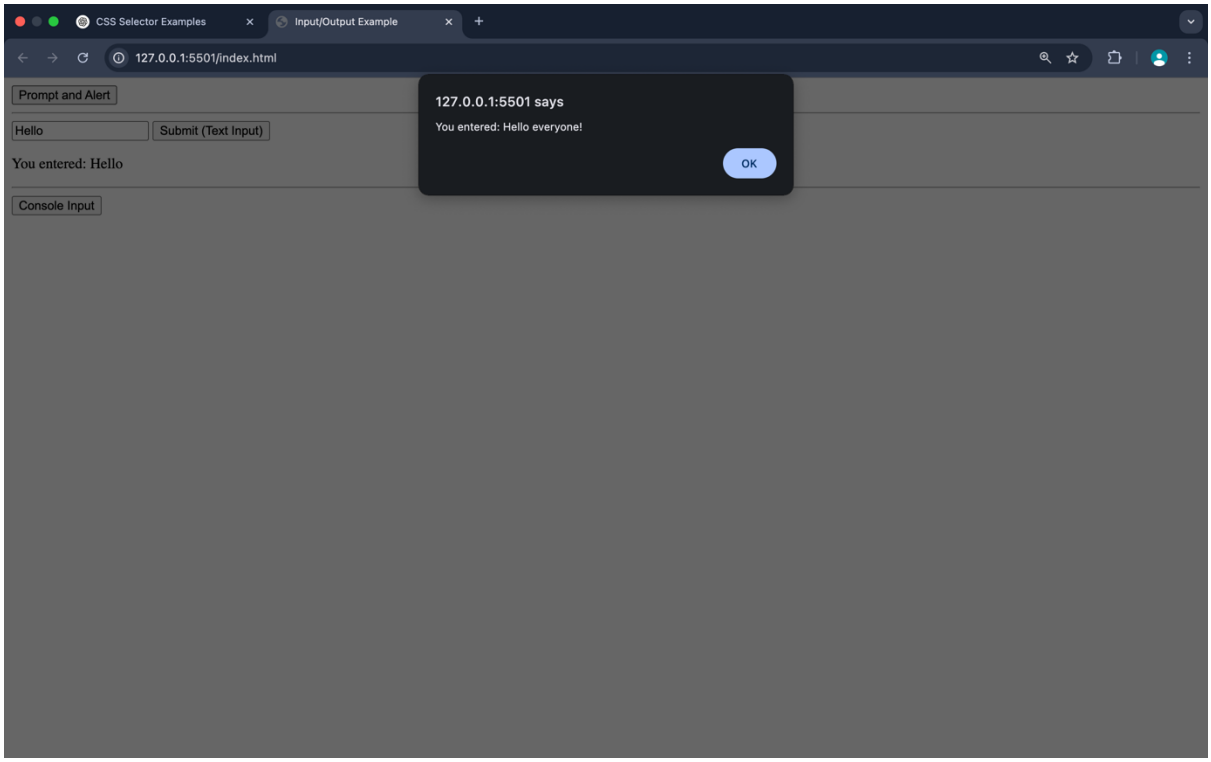
These methods help interact with the user and provide a seamless experience for web applications. Here's an example of how to use these techniques together in a program.

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Input/Output Example</title>
  <script>
    function promptAlert() {
      alert("You entered: " + prompt("Enter something:"));
    }
    function textInput() {
      document.getElementById("output").innerText = "You entered: " +
document.getElementById("inputField").value;
    }
    function consoleInput() {
      console.log("You entered: " + prompt("Enter a number:"));
    }
  </script>
</head>
<body>

  <button onclick="promptAlert()">Prompt and Alert</button>
  <hr>
  <input type="text" id="inputField">
  <button onclick="textInput()">Submit (Text Input)</button>
  <p id="output"></p>
  <hr>
  <button onclick="consoleInput()">Console Input</button>

</body>
</html>
```

OUTPUT:


CSS Selector Examples
Input/Output Example
127.0.0.1:5501/index.html
Prompt and Alert
Hello
Submit (Text Input)
You entered: Hello
Console Input


CSS Selector Examples
Input/Output Example
127.0.0.1:5501/index.html
Prompt and Alert
Hello
Submit (Text Input)
You entered: Hello
Console Input
127.0.0.1:5501 says
You entered: Hello everyone!
OK

C. Create a webpage which uses prompt dialogue box to ask a voter for his name and age.Display the information in table format along with either the voter can vote or not

Introduction

This program demonstrates how to use JavaScript to take input from the user through a prompt dialog box. In this case, we will ask the user for their name and age to check if they are eligible to vote. Based on the input, the program will determine whether the user meets the voting age requirement (18 years or older) and will display this information in a table format. This example highlights how to interact with users and display dynamic content on a webpage using JavaScript.

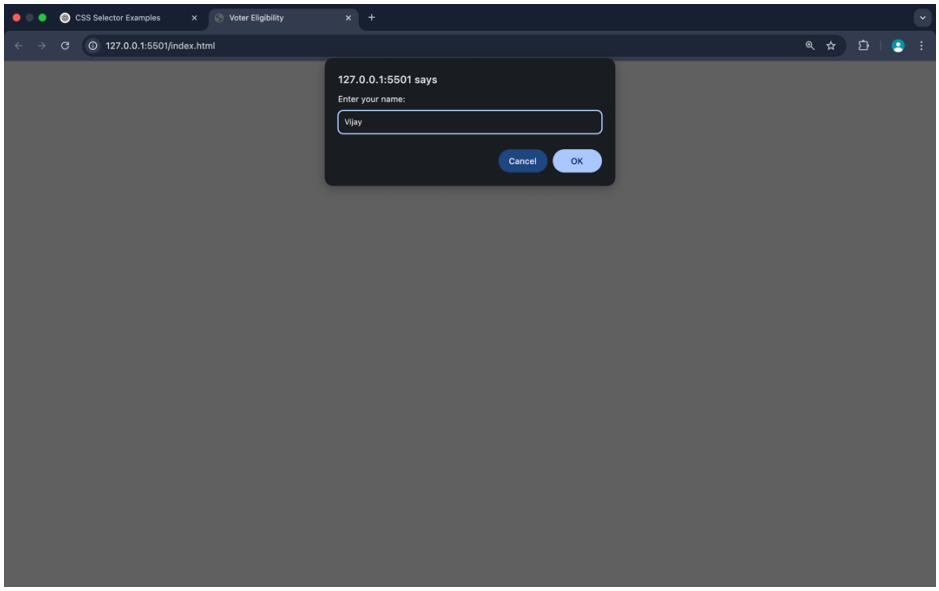
PROGRAMME:

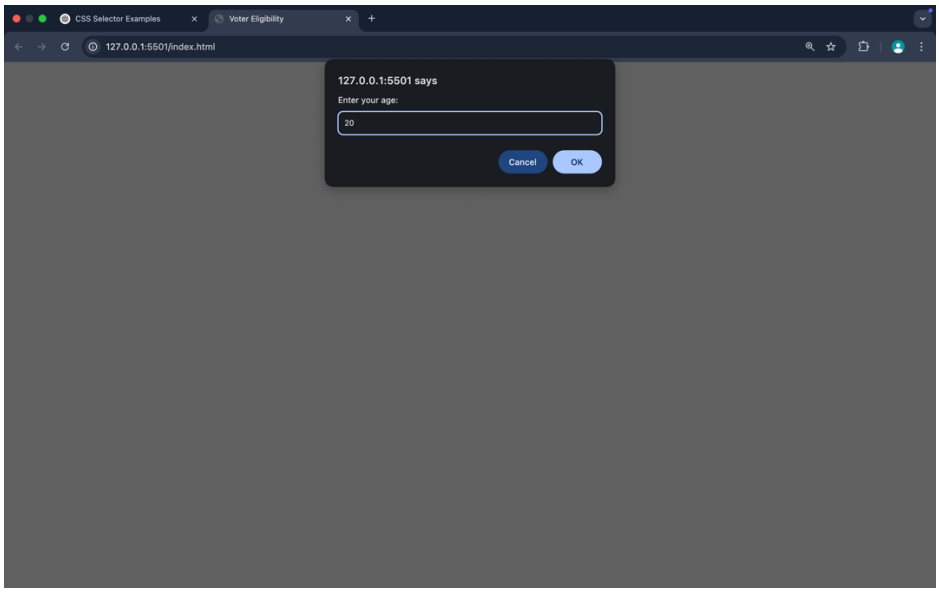
```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Voter Eligibility</title>
 <style>
  body {
   font-family: Arial, sans-serif;
   background-color: #f0f0f0;
   text-align: center;
   padding: 20px;
  }
  h1 {
   color: #333;
   margin-bottom: 20px;
  }
  button {
   background-color: #4CAF50;
   color: white;
   padding: 10px 20px;
   border: none;
```

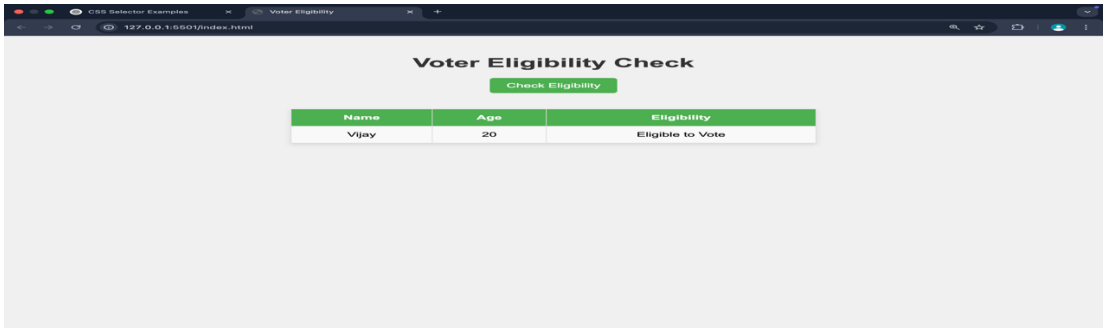
```
    cursor: pointer;
    font-size: 16px;
    border-radius: 5px;
    margin-bottom: 20px;
  }
  button:hover {
    background-color: #45a049;
  }
  table {
    margin: 20px auto;
    border-collapse: collapse;
    width: 50%;
    background-color: #fff;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  }
  th, td {
    border: 1px solid #ddd;
    padding: 12px;
    text-align: center;
  }
  th {
    background-color: #4CAF50;
    color: white;
  }
  td {
    background-color: #f9f9f9;
  }
</style>
<script>
  function checkVoterEligibility() {
    let name = prompt("Enter your name:");
    let age = prompt("Enter your age:");
```

```
    // Convert age to a number
    age = Number(age);
    let eligibility = (age >= 18) ? "Eligible to Vote" : "Not Eligible to Vote";

    // Display the information in a table
    let table = `<table>
                <tr><th>Name</th><th>Age</th><th>Eligibility</th></tr>
                <tr><td>${name}</td><td>${age}</td><td>${eligibility}</td></tr>
               </table>`;

    document.getElementById("voterInfo").innerHTML = table;
   }
  </script>
</head>
<body>

  <h1>Voter Eligibility Check</h1>
  <button onclick="checkVoterEligibility()">Check Eligibility</button>
  <div id="voterInfo"></div>

</body>
</html>
```

OUTPUT:


CSS Selector Examples
Voter Eligibility
127.0.0.1:5501/index.html
127.0.0.1:5501 says
Enter your name:
Vijay
Cancel
OK


CSS Selector Examples
Voter Eligibility
127.0.0.1:5501/index.html
127.0.0.1:5501 says
Enter your age:
20
Cancel
OK


Voter Eligibility Check
Check Eligibility
Name
Age
Eligibility
Vijay
20
Eligible to Vote

Exercise-6.JavaScript Pre-defined and User-defined Objects

A.Write a program using document object properties and methods.

Introduction

In JavaScript, the Document Object Model (DOM) represents the HTML document as a tree structure where each node is an object. The `document` object is a pre-defined object in JavaScript that allows access to and manipulation of the elements in a web page. This program will demonstrate how to use various `document` object properties and methods, such as `getElementById`, `innerHTML`, and `createElement`, to interact with and modify the content of an HTML document dynamically. By leveraging the `document` object, JavaScript can alter the content, style, and structure of a webpage on the fly, enhancing user interaction and experience.

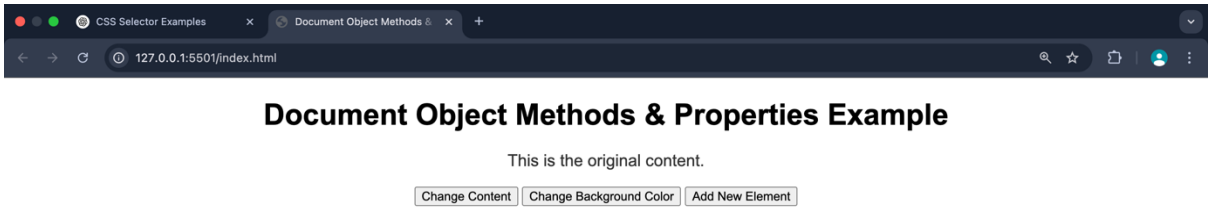
PROGRAMME:

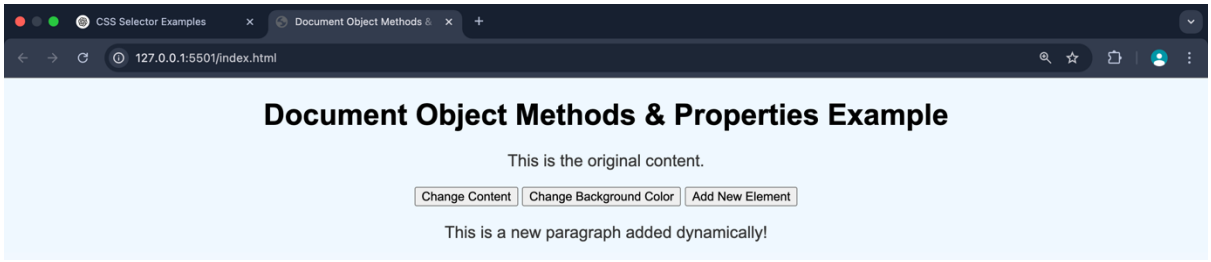
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document Object Demo</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin: 50px;
    }
    #output {
      margin-top: 20px;
      font-size: 20px;
      color: blue;
    }
    button {
```

```
            padding: 10px 15px;
            font-size: 16px;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <h1 id="title">Hello, World!</h1>
    <button id="changeTextBtn">Change Text</button>
    <button id="addParaBtn">Add Paragraph</button>
    <div id="output"></div>
    <script>
        // Modify document properties
        document.title = "Updated Title Using JavaScript";
        document.body.style.backgroundColor = "#f0f8ff"; // Light blue background
        // Get elements
        const title = document.getElementById("title");
        const changeTextBtn = document.getElementById("changeTextBtn");
        const addParaBtn = document.getElementById("addParaBtn");
        const outputDiv = document.getElementById("output");
        // Change text content on button click
        changeTextBtn.addEventListener("click", function () {
            title.textContent = "Text Changed!";
            outputDiv.innerHTML = "<strong>Document object properties are
powerful!</strong>";
        });
        // Add a new paragraph dynamically
        addParaBtn.addEventListener("click", function () {
            const newPara = document.createElement("p");
            newPara.textContent = "This is a dynamically added paragraph.";
```

```
        newPara.style.color = "green";
        document.body.appendChild(newPara);
    });
    // Log document properties
    console.log("Document URL:", document.URL);
    console.log("Document Title:", document.title);
</script>

</body>
</html>
```

OUTPUT:





B. Write a program using window object properties and methods.

Introduction

The `window` object in JavaScript serves as the global object that represents the browser window in which the code is executed. It provides properties and methods for manipulating the browser window, accessing the DOM, and interacting with features like alerts, prompts, navigation, and timers. The `window` object is fundamental in web development as it enables developers to control various aspects of the user interface and behavior.

Some commonly used properties and methods of the `window` object include:

1. **Properties**: `window.innerWidth`, `window.innerHeight`, `window.location`, `window.document`.
2. **Methods**: `window.alert()`, `window.prompt()`, `window.open()`, `window.close()`, `window.setTimeout()`, `window.clearTimeout()`, etc.

Below is a program demonstrating the usage of several `window` object properties and methods.

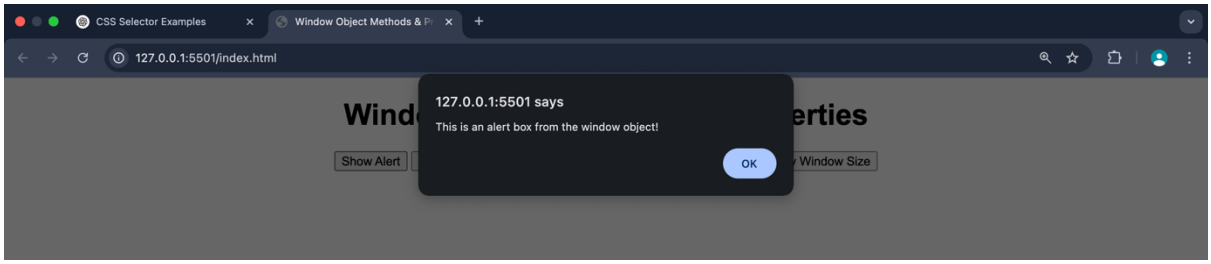
PROGRAMME:

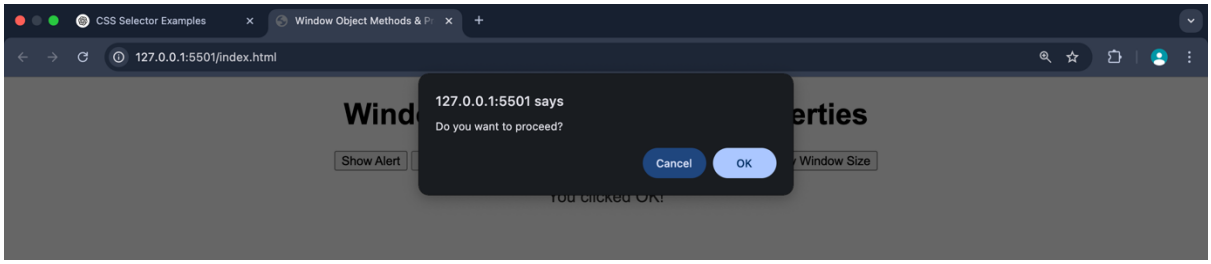
```
<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

 <title>Window Object Methods & Properties</title>

 <style>

  body {

   font-family: Arial, sans-serif;

   text-align: center;

   margin-top: 20px;

  }

  p {

   font-size: 18px;

   color: #333;

  }
```

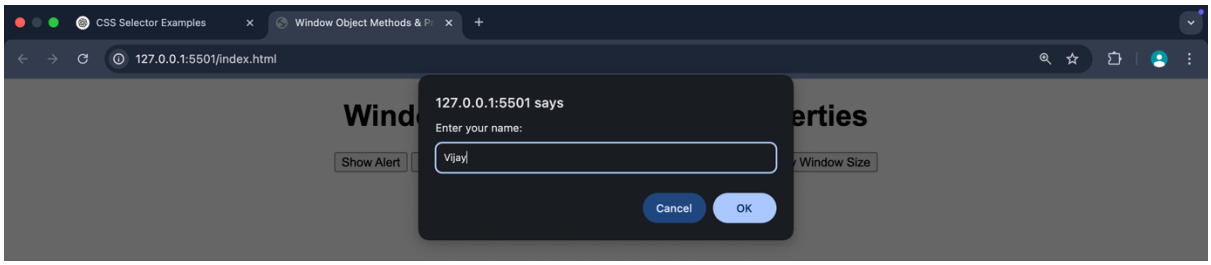
```
  </style>
  <script>
    function showAlert() {
      // Using window.alert to display an alert box
      window.alert("This is an alert box from the window object!");
    }
    function showConfirmation() {
      // Using window.confirm to show a confirmation dialog box
      let result = window.confirm("Do you want to proceed?");
      document.getElementById("confirmationResult").innerHTML = result ? "You clicked OK!" : "You clicked
Cancel!";
    }
    function showPrompt() {
      // Using window.prompt to ask for user input
      let name = window.prompt("Enter your name:");
      document.getElementById("userName").innerHTML = "Hello, " + name + "!";
    }
    function openNewWindow() {
      // Using window.open to open a new window or tab
      window.open("https://www.example.com", "_blank");
    }
    function displayWindowSize() {
      // Using window.innerWidth and window.innerHeight to get window size
      let width = window.innerWidth;
      let height = window.innerHeight;
      document.getElementById("windowSize").innerHTML = "Window size: " + width + "x" + height;
    }
  </script>
```

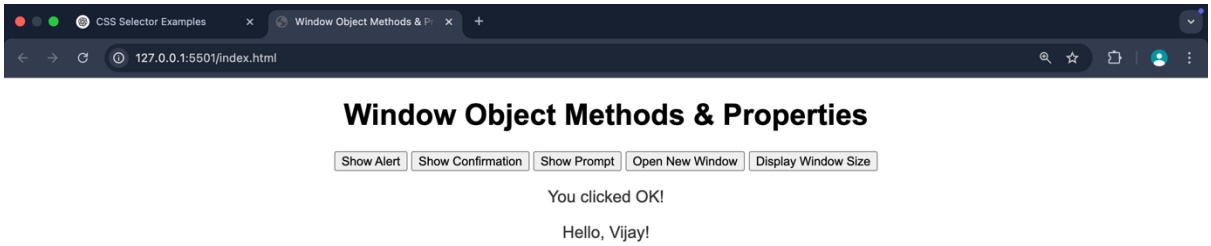
```
</head>
<body>
  <h1>Window Object Methods & Properties</h1>
  <button onclick="showAlert()">Show Alert</button>
  <button onclick="showConfirmation()">Show Confirmation</button>
  <button onclick="showPrompt()">Show Prompt</button>
  <button onclick="openNewWindow()">Open New Window</button>
  <button onclick="displayWindowSize()">Display Window Size</button>
  <p id="confirmationResult"></p>
  <p id="userName"></p>
  <p id="windowSize"></p>
</body>
</html>
```

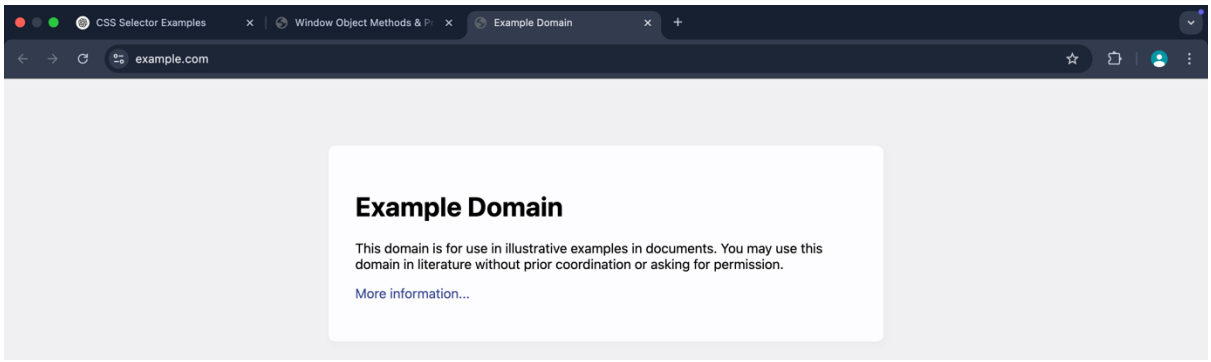
OUTPUT:

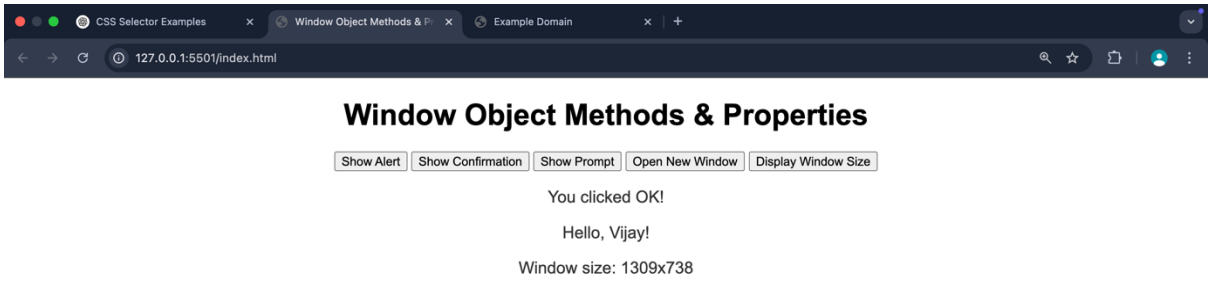












C.Write a program using math object properties and methods.

INTRODUCTION:

The Math object in JavaScript serves as a powerful tool for performing a wide range of mathematical operations and computations. It is equipped with a variety of methods and properties that simplify complex calculations, making it an essential component for developers working with numbers and mathematical logic. This object enables tasks such as rounding values to the nearest integer, raising numbers to specific powers, extracting square roots, and accessing important mathematical constants like Pi (π) and Euler's number (e). Additionally, it provides the ability to generate random numbers, which are widely used in simulations, games, and other applications requiring unpredictability.

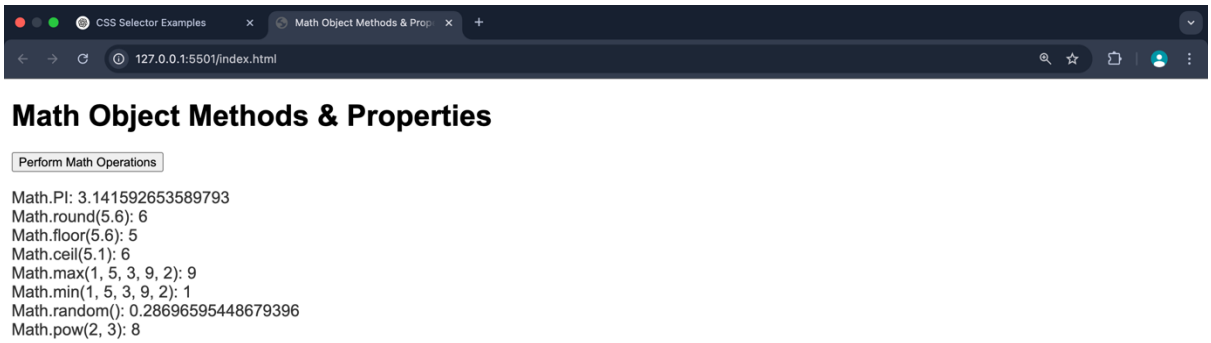
By incorporating the Math object into programs, developers can ensure accuracy and efficiency in handling numerical operations. Below, a demonstration program showcases some of the key functionalities provided by the Math object, highlighting its versatility and importance in JavaScript programming.

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Math Object Methods & Properties</title>
 <style>
  body {
   font-family: Arial, sans-serif;
   text-align: left;
   margin-top: 20px;
  }
  p {
   font-size: 18px;
   color: #333;
  }
 </style>
 <script>
  function performMathOperations() {
   let piValue = Math.PI;
   let roundedValue = Math.round(5.6);
   let flooredValue = Math.floor(5.6);
   let ceiledValue = Math.ceil(5.1);
   let maxValue = Math.max(1, 5, 3, 9, 2);
   let minValue = Math.min(1, 5, 3, 9, 2);
   let randomValue = Math.random();
   let powerValue = Math.pow(2, 3); // 2^3 = 8
   document.getElementById("result").innerHTML = `
    Math.PI: ${piValue}<br>
    Math.round(5.6): ${roundedValue}<br>
    Math.floor(5.6): ${flooredValue}<br>
    Math.ceil(5.1): ${ceiledValue}<br>
```

```
      Math.max(1, 5, 3, 9, 2): ${maxValue}<br>
      Math.min(1, 5, 3, 9, 2): ${minValue}<br>
      Math.random(): ${randomValue}<br>
      Math.pow(2, 3): ${powerValue}
    `;
  }
 </script>
</head>
<body>
 <h1>Math Object Methods & Properties</h1>
 <button onclick="performMathOperations()">Perform Math Operations</button>
 <p id="result"></p>
</body>
</html>
```

Output:



D.Write a program using string object properties and methods.

Introduction

The String object in JavaScript is a powerful tool that provides various properties and methods to handle and manipulate text effectively. By using these features, developers can perform operations such as retrieving the length of a string, converting its case, extracting substrings, searching for specific characters, replacing text, and splitting strings into arrays. These functionalities are essential in many programming tasks involving text processing and manipulation.

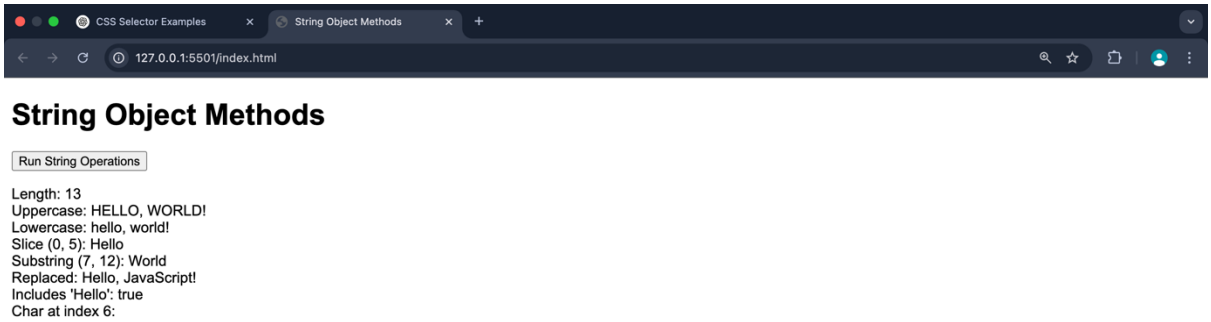
Below is a demonstration program that showcases the use of String object properties and methods in a structured HTML format.

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>String Object Methods</title>
  <style>
    body { font-family: Arial, sans-serif; text-align: center; margin-top: 20px; }
  </style>
</head>
<body>
  <h1>String Object Methods</h1>
  <button onclick="performStringOperations()">Run String Operations</button>
  <p id="result"></p>
  <script>
    function performStringOperations() {
      let str = "Hello, World!";
      let result = `
        Length: ${str.length} <br>
        Uppercase: ${str.toUpperCase()} <br>
        Lowercase: ${str.toLowerCase()} <br>
        Slice (0, 5): ${str.slice(0, 5)} <br>
        Substring (7, 12): ${str.substring(7, 12)} <br>
        Replaced: ${str.replace("World", "JavaScript")} <br>
        Includes 'Hello': ${str.includes("Hello")} <br>
        Char at index 6: ${str.charAt(6)} <br ;
```

```
    document.getElementById("result").innerHTML = result;
  }
 </script>
</body>
</html>
```

OUTPUT:



E.Write a program using regex object properties and methods.

Introduction

Regular Expressions (Regex) are a powerful feature in JavaScript used for pattern matching and text manipulation. The `RegExp` object provides various properties and methods to search, test, match, and replace strings based on specified patterns. Regex is widely utilized in applications such as validating input, extracting specific substrings, formatting text, and parsing data efficiently.

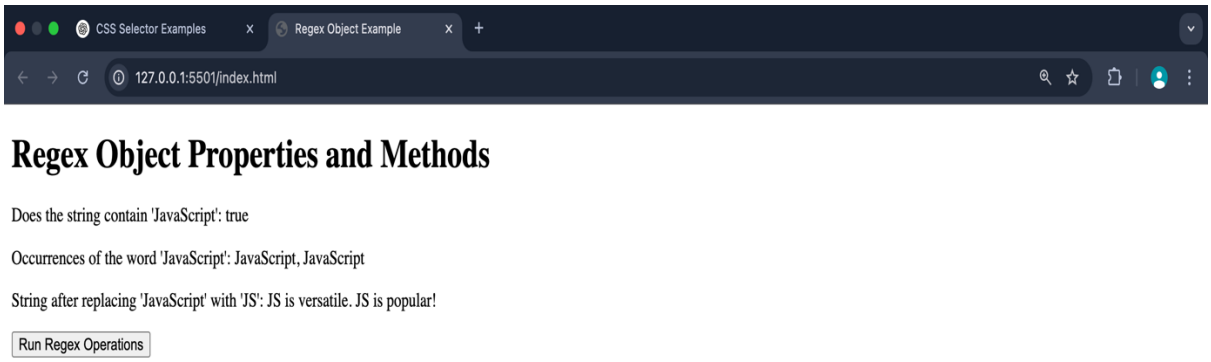
Below is a demonstration program showcasing the use of Regex object properties and methods in a structured HTML format.

PROGRAMME:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Regex Object Example</title>
</head>
```

```
<body>
    <h1>Regex Object Properties and Methods</h1>
    <p id="testResult">Does the string contain 'JavaScript': </p>
    <p id="matchResult">Occurrences of the word 'test': </p>
    <p id="replaceResult">String after replacing 'JavaScript' with 'JS': </p>
    <button onclick="performRegexOperations()">Run Regex Operations</button>
    <script>
        function performRegexOperations() {
            // Define a string for pattern matching
            let str = "JavaScript is versatile. JavaScript is popular!";
            let regex = /JavaScript/g;
            let containsJavaScript = regex.test(str);
            document.getElementById("testResult").innerHTML =
                "Does the string contain 'JavaScript': " + containsJavaScript;
            let matches = str.match(regex);
            document.getElementById("matchResult").innerHTML =
                "Occurrences of the word 'JavaScript': " + matches.join(", ")
            let replacedString = str.replace(regex, "JS");
            document.getElementById("replaceResult").innerHTML =
                "String after replacing 'JavaScript' with 'JS': " + replacedString;
        }
    </script>
</body>
</html>
```

OUTPUT:



F.Write a program using date object properties and methods.

Introduction

The Date object in JavaScript is a versatile tool that allows developers to work with dates and times. It provides methods to create, modify, and retrieve information about dates and times, as well as perform calculations involving date and time intervals. Developers can use the Date object for tasks such as displaying the current date and time, extracting individual components like the year, month, and day, formatting dates, and performing date arithmetic.

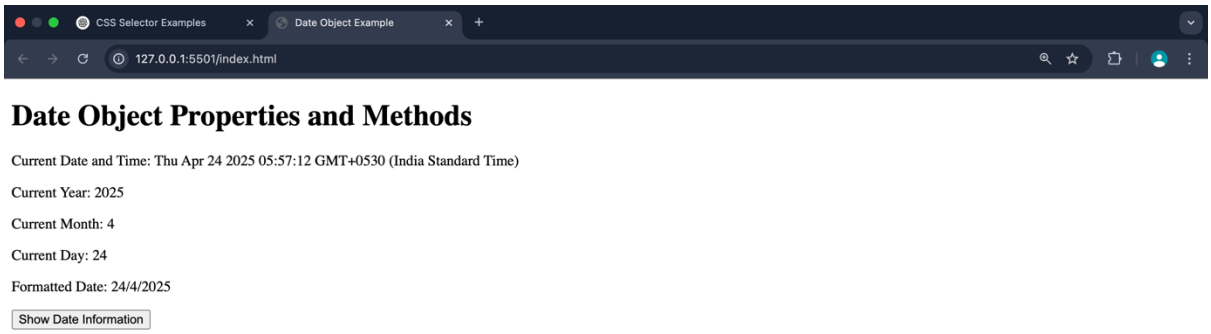
Below is a program demonstrating the use of various properties and methods of the Date object, presented in a structured HTML format.

PROGRAMME:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Date Object Example</title>

</head>

<body>

    <h1>Date Object Properties and Methods</h1>

    <p id="currentDate">Current Date and Time: </p>

    <p id="yearResult">Current Year: </p>
```

```
    <p id="monthResult">Current Month: </p>
    <p id="dayResult">Current Day: </p>
    <p id="formattedDate">Formatted Date: </p>
    <button onclick="displayDateInfo()">Show Date Information</button>
    <script>
        function displayDateInfo() {
            let currentDate = new Date();
            document.getElementById("currentDate").innerHTML =
                "Current Date and Time: " + currentDate;
            let year = currentDate.getFullYear();
            document.getElementById("yearResult").innerHTML =
                "Current Year: " + year;
            let month = currentDate.getMonth() + 1;
            document.getElementById("monthResult").innerHTML =
                "Current Month: " + month;
            let day = currentDate.getDate();
            document.getElementById("dayResult").innerHTML =
                "Current Day: " + day;
            let formattedDate = day + "/" + month + "/" + year;
            document.getElementById("formattedDate").innerHTML =
                "Formatted Date: " + formattedDate;
        }
    </script>
</body>
</html>
```

OUTPUT:


CSS Selector Examples
Date Object Example
127.0.0.1:5501/index.html
Date Object Properties and Methods
Current Date and Time: Thu Apr 24 2025 05:57:12 GMT+0530 (India Standard Time)
Current Year: 2025
Current Month: 4
Current Day: 24
Formatted Date: 24/4/2025
Show Date Information

Exercise-7.JavaScript Conditional Statements and Loops

A. Write a program which asks the user to enter three integers, obtains the numbers from the user and outputs HTML text that displays the larger number followed by the words "LARGER NUMBER" in an information message dialog. If the numbers are equal, output HTML text as "EQUAL NUMBERS".

Introduction

Conditional statements and loops are key features in JavaScript, allowing developers to control the flow of execution based on conditions and repetitive tasks. In this example, we will write a program where the user is prompted to enter three integers. The program evaluates these inputs and displays a message stating the largest number along with the text "LARGER NUMBER." If all the numbers are equal, the program will instead display the message "EQUAL NUMBERS."

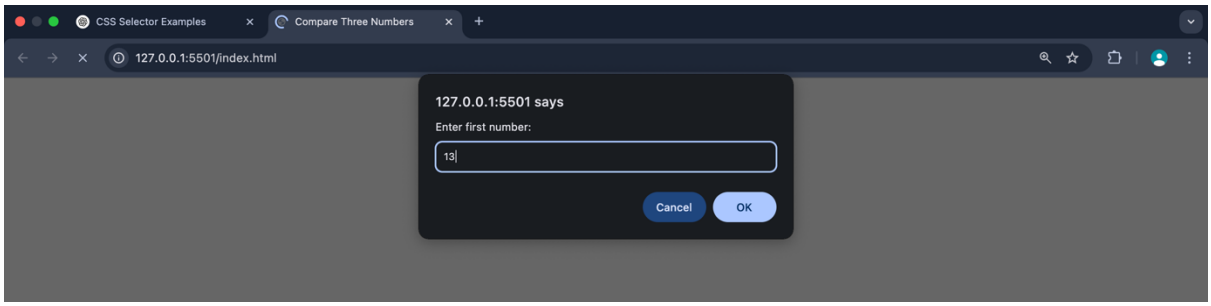
Below is the complete program in HTML and JavaScript:

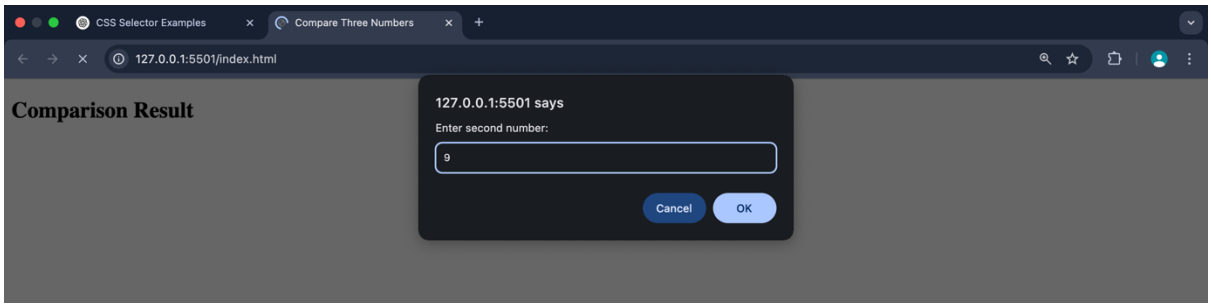
PROGRAMME:

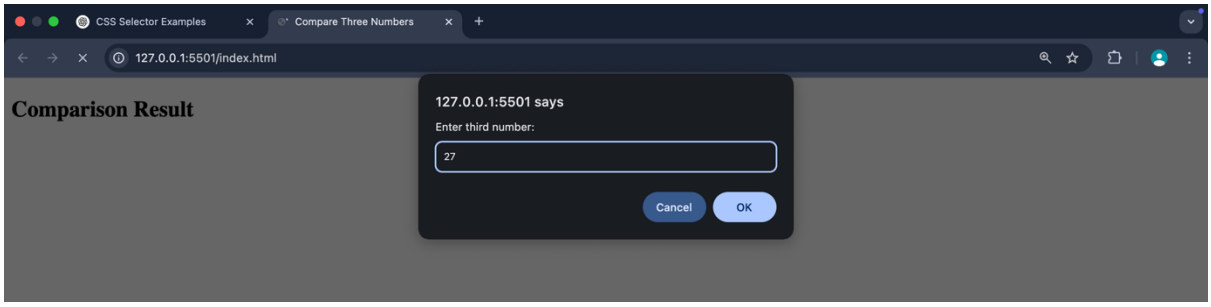
```
<!DOCTYPE html>
<html>
<head>
 <title>Compare Three Numbers</title>
 <script>
  function compareNumbers() {
   let a = parseInt(prompt("Enter first number:"));
   let b = parseInt(prompt("Enter second number:"));
   let c = parseInt(prompt("Enter third number:"));
   let msg = "";
   if (a === b && b === c) {
    msg = "EQUAL NUMBERS";
   } else {
    let max = Math.max(a, b, c);
    msg = max + " LARGER NUMBER";
   }
   document.getElementById("result").innerHTML = `<b>${msg}</b>`;
   alert(msg); // optional info dialog
  }
```

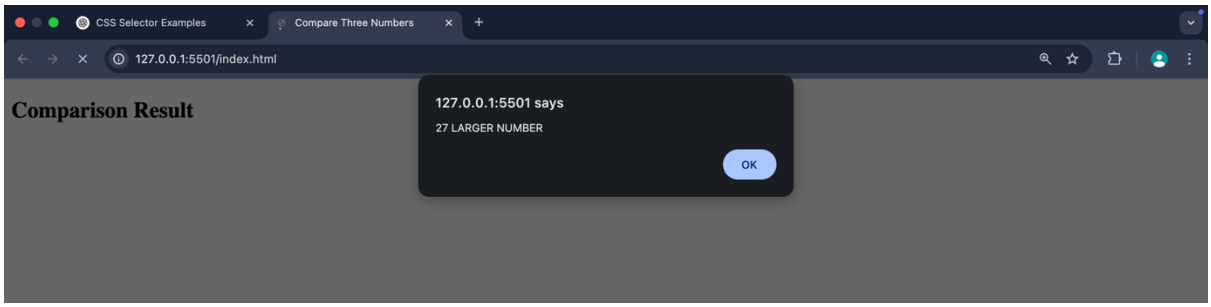
```
 </script>
</head>
<body onload="compareNumbers()">
 <h2>Comparison Result</h2>
 <p id="result" style="color: green;"></p>
</body>
</html
```

OUTPUT:









B. Write a program to display week days using switch case.

Introduction

The `switch` statement in JavaScript is a powerful tool for performing conditional operations based on different cases. It allows developers to execute specific blocks of code depending on the value of an expression. In this example, we will create a program that prompts the user to input a number (1–7) representing the day of the week, and then displays the corresponding weekday using the `switch` statement.

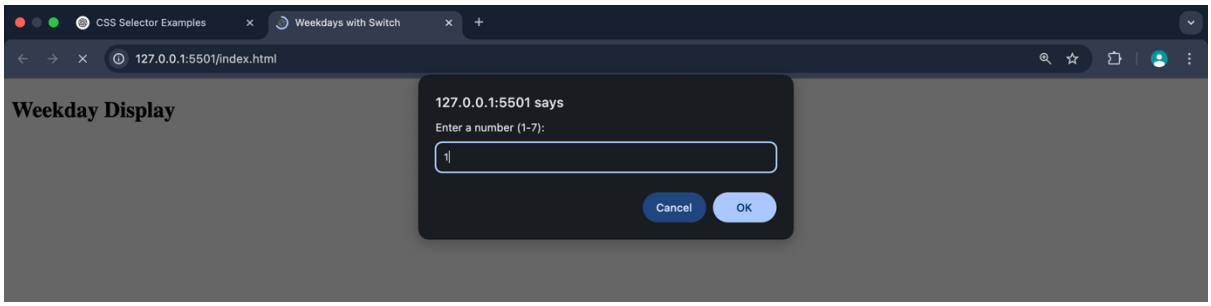
Below is the complete program in HTML and JavaScript:

PROGRAMME:

```
<!DOCTYPE html>

<html>

<head>

 <title>Weekdays with Switch</title>

 <script>

  function showDay() {

   let num = parseInt(prompt("Enter a number (1-7):"));

   let day;

   switch (num) {

    case 1: day = "Sunday"; break;

    case 2: day = "Monday"; break;

    case 3: day = "Tuesday"; break;

    case 4: day = "Wednesday"; break;

    case 5: day = "Thursday"; break;

    case 6: day = "Friday"; break;

    case 7: day = "Saturday"; break;

    default: day = "Invalid input!";

   }

   document.getElementById("output").textContent = day;

  }

 </script>
```

```
</head>
<body onload="showDay()">
  <h2>Weekday Display</h2>
  <p id="output" style="font-weight: bold; color: blue;"></p>
</body>
</html>
```

OUTPUT:



C. Write a program to print 1 to 10 numbers using for, while and do-while loops

INTRODUCTION:

Loops are an integral feature of JavaScript, enabling developers to handle repetitive tasks efficiently and systematically. By using looping structures such as `for`, `while`, and `do-while` loops, programmers can iterate over a sequence of values, perform computations, or automate repetitive actions in a predictable and controlled manner. These loops differ in their initialization, condition checks, and execution processes, making them suitable for various scenarios. The `for` loop is particularly useful when the number of iterations is predefined, whereas the `while` loop excels in cases where the condition must be evaluated before execution. The `do-while` loop ensures that the code is executed at least once, even if the condition is initially false.

In this example, we will demonstrate how to use these three looping constructs to print numbers from 1 to 10. The program provides a clear illustration of their differences and how they can be applied effectively in practical coding tasks. Below is the complete implementation presented in a structured HTML and JavaScript format.

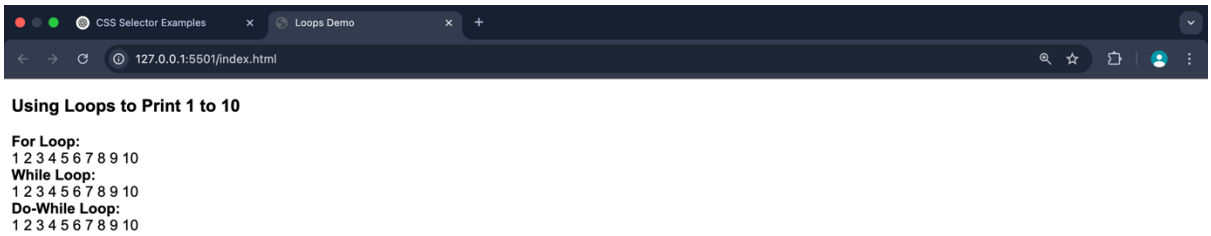
PROGRAMME:

```
<!DOCTYPE html>

<html>

<head>

  <title>Loops Demo</title>

  <style>body { font-family: Arial; }</style>

</head>

<body>

<h3>Using Loops to Print 1 to 10</h3>

<div id="output"></div>

<script>

  let result = "<b>For Loop:</b><br>";

  for (let i = 1; i <= 10; i++) result += i + " ";

  result += "<br><b>While Loop:</b><br>";

  let j = 1;

  while (j <= 10) result += j++ + " ";

  result += "<br><b>Do-While Loop:</b><br>";

  let k = 1;

  do {

    result += k++ + " ";

  } while (k <= 10);

  document.getElementById("output").innerHTML = result;

</script>
```

</body>

</html>

OUTPUT:



D. Write a program to print data in object using for-in, for-each and for-of loops

Introduction

In JavaScript, iterating over the properties or values of an object is a common requirement, often achieved using different looping constructs. The `for-in` loop is specifically designed for enumerating the keys or properties of an object. The `for-each` method is used to iterate over the elements of arrays or other iterable objects, and it allows for the execution of a function on each element. The `for-of` loop, on the other hand, is ideal for iterating directly over the values of iterable objects such as arrays, strings, or sets.

In this example, we will demonstrate how to use these loops effectively to iterate over an object's data. The implementation highlights their usage and applicability in dealing with object properties and values. Below is the complete program in HTML and JavaScript format.

PROGRAMME:

<!DOCTYPE html>

<html>

<head>

<title>Looping Through Object</title>

<style>body { font-family: Arial; }</style>

</head>

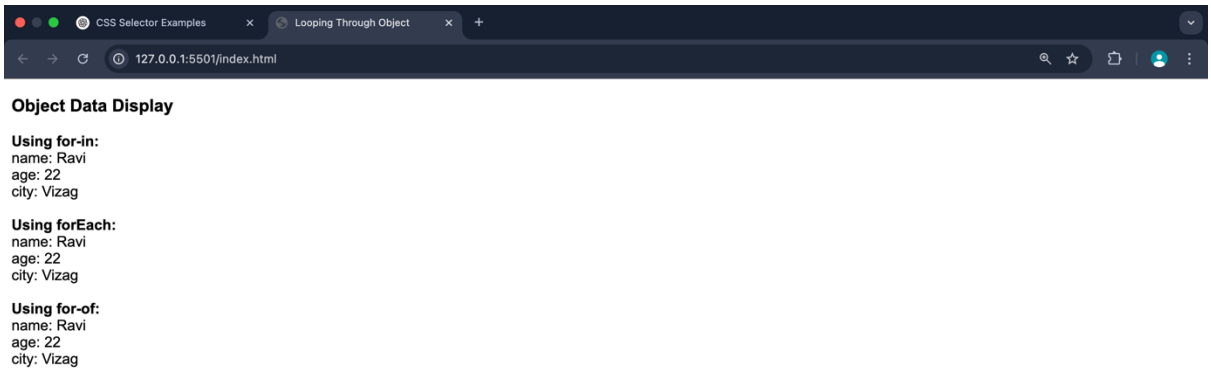
<body>

<h3>Object Data Display</h3>

<script>

```
const person = { name: "Ravi", age: 22, city: "Vizag" };
let result = "<b>Using for-in:</b><br>";

for (let key in person) {
  result += `${key}: ${person[key]}<br>`;
}

// Convert object to array for forEach and for-of
const entries = Object.entries(person);
result += "<br><b>Using forEach:</b><br>";
entries.forEach(([key, value]) => result += `${key}: ${value}<br>`);

result += "<br><b>Using for-of:</b><br>";
for (let [key, value] of entries) {
  result += `${key}: ${value}<br>`;
}
document.getElementById("output").innerHTML = result;
</script>
</body>
</html>
```

OUTPUT:



E. Develop a program to determine whether a given number is an 'ARMSTRONG NUMBER' or not. (Eg: 153 is an Armstrong number, since sum of the cube of the digits is equal to the number i.e., 13+ 53+ 33=153).

Introduction

An Armstrong number, also known as a Narcissistic number, is a number that satisfies a unique mathematical property. Specifically, for a number with (n) digits, it is considered an Armstrong number if the sum of each digit raised to the power of (n) equals the number itself. This property can be applied to numbers with any number of digits. For example:

- The number **153** is an Armstrong number because (13 + 3^3 = 153).
- The number **9474** is also an Armstrong number because (94 + 74 = 9474).

Armstrong numbers have applications in mathematical theory and computational problems involving digit-based calculations. In this program, we will implement a solution to check whether a user-provided number satisfies the Armstrong number condition. The program is designed to be interactive and will use iteration to calculate the required sum of powers.

PROGRAMME:

```
<!DOCTYPE html>

<html>

<head>

 <title>Armstrong Number Checker</title>

 <style>

  body { font-family: Arial; }

  #result { font-weight: bold; color: green; }

 </style>

</head>

<body>

<h3>Armstrong Number Checker</h3>

<input type="number" id="numInput" placeholder="Enter a number">

<button onclick="checkArmstrong()">Check</button>

<p id="result"></p>


<script>

 function checkArmstrong() {
```

```
  const num = document.getElementById("numInput").value;
  const digits = num.split('').map(Number);
  const power = digits.length;
  const sum = digits.reduce((acc, d) => acc + Math.pow(d, power), 0);

  document.getElementById("result").textContent =
    sum == num ? `${num} is an Armstrong Number` : `${num} is NOT an Armstrong Number`;
 }
</script>

</body>
</html>
```

OUTPUT:

F. Write a program to display the denomination of the amount deposited in the bank in terms of 100's, 50's, 20's, 10's, 5's, 2's& 1's. (Eg: If deposited amount is Rs.163, the output should be 1-100's, 1-50's, 1- 10's, 1-2's & 1- 1's)

Introduction

Currency denominations are an essential part of financial calculations, especially for banking applications. When a specific amount is deposited into the bank, it can be represented in terms of denominations such as 100's, 50's, 20's, 10's, 5's, 2's, and 1's. Determining the denomination breakdown helps identify the quantity of each currency note required to make up the given amount, ensuring clarity in transactions.

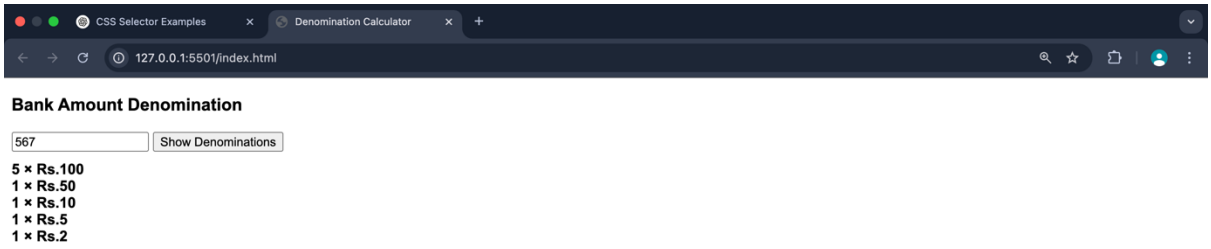
The following program takes a deposited amount from the user and computes the count of each denomination required to represent it. The calculations are performed iteratively by using division and modulo operations. This program provides an intuitive way to handle financial computations involving denominations.

PROGRAMME:

```
<!DOCTYPE html>
<html>
<head>
 <title>Denomination Calculator</title>
 <style>
  body { font-family: Arial; }
  #result { margin-top: 10px; font-weight: bold; }
 </style>
</head>
<body>
<h3>Bank Amount Denomination</h3>
<input type="number" id="amount" placeholder="Enter amount in Rs">
<button onclick="calculateDenomination()">Show Denominations</button>
<div id="result"></div>
<script>
 function calculateDenomination() {
  let amt = parseInt(document.getElementById("amount").value);
  const notes = [100, 50, 20, 10, 5, 2, 1];
```

```
    let output = "";
    for (let note of notes) {
      let count = Math.floor(amt / note);
      if (count > 0) {
        output += `${count} × Rs.${note}<br>`;
        amt %= note;
      }
    }
    document.getElementById("result").innerHTML = output;
  }
</script>
</body>
</html>
```

OUTPUT:



Exercise-8.Java Script Functions and Events

a) Design a HTML having a text box and four buttons named Factorial, Fibonacci, Prime, and Palindrome. When a button is pressed an appropriate function should be called to display i. Factorial of that number ii. Fibonacci series up to that number iii. Prime numbers up to that number iv. Is it palindrome or not

Introduction

JavaScript functions and events are fundamental for creating interactive web applications. Functions encapsulate reusable logic, while events serve as triggers for specific actions based on user interactions. In this example, we design an HTML page with a text box and four buttons labeled **Factorial**, **Fibonacci**, **Prime**, and **Palindrome**. Each button is associated with a corresponding function to perform computations and display results dynamically.

The functionalities implemented include:

1. **Factorial Calculation**: Computes the factorial of the input number.
2. **Fibonacci Series**: Generates the Fibonacci sequence up to the input number.
3. **Prime Numbers**: Lists all prime numbers up to the input number.
4. **Palindrome Check**: Determines whether the input number is a palindrome.

The following code demonstrates how functions and events are integrated seamlessly into the HTML design.

PROGRAMME:

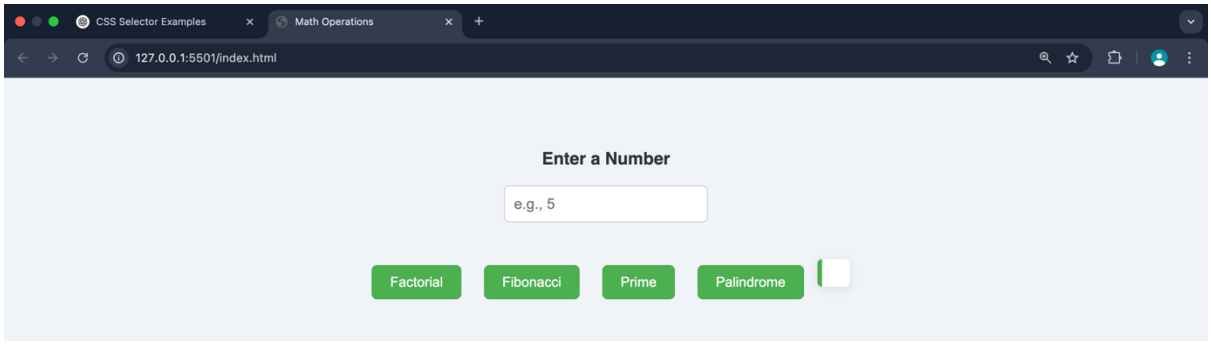
```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Math Operations</title>
 <style>
  body {
   font-family: 'Segoe UI', sans-serif;
   background: #f0f4f8;
   text-align: center;
   padding: 50px;
  }
  h3 {
   color: #333;
```

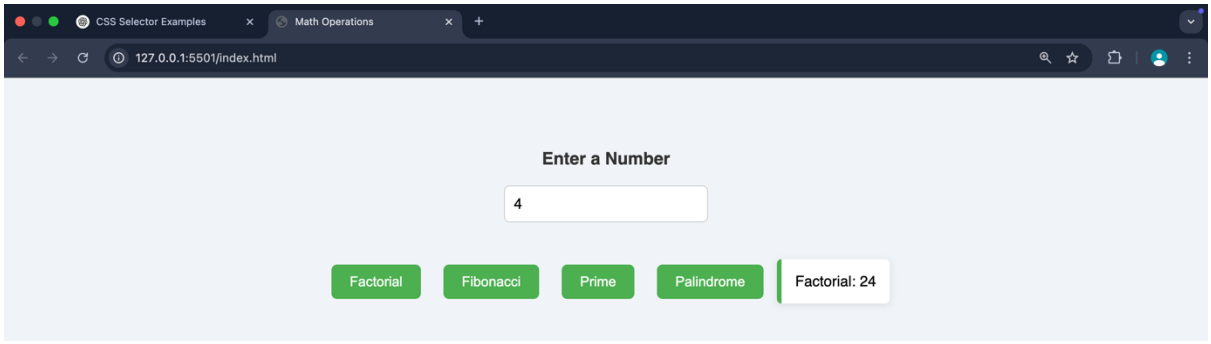
```
}
input {
  padding: 10px;
  width: 200px;
  font-size: 16px;
  margin-bottom: 20px;
  border: 1px solid #ccc;
  border-radius: 6px;
}
button {
  padding: 10px 20px;
  margin: 10px;
  font-size: 15px;
  border: none;
  border-radius: 6px;
  background-color: #4CAF50;
  color: white;
  cursor: pointer;
  transition: background 0.3s;
}
button:hover {
  background-color: #45a049;
}
#result {
  margin-top: 20px;
  padding: 15px;
  background-color: #fff;
  border-left: 5px solid #4CAF50;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
  display: inline-block;
  text-align: left;
  max-width: 400px;
  border-radius: 5px;
```

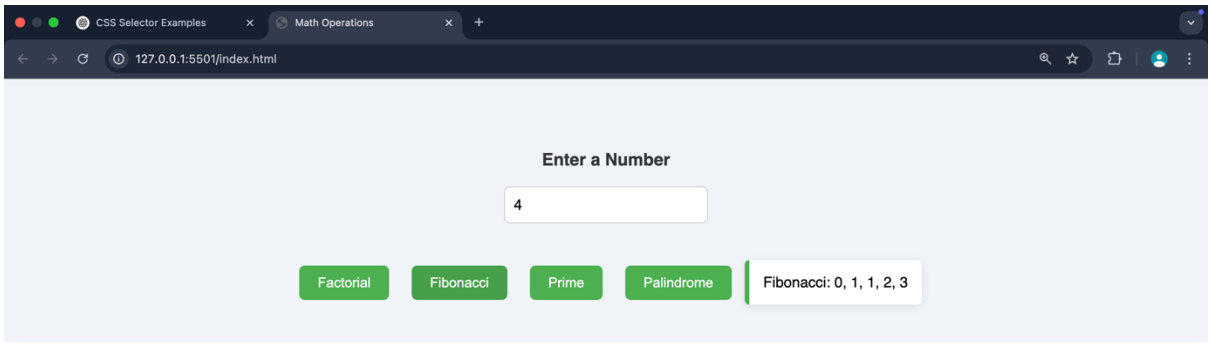
```
    font-size: 16px;
   }
  </style>
</head>
<body>

  <h3>Enter a Number</h3>
  <input type="number" id="num" placeholder="e.g., 5"><br>
  <button onclick="factorial()">Factorial</button>
  <button onclick="fibonacci()">Fibonacci</button>
  <button onclick="prime()">Prime</button>
  <button onclick="palindrome()">Palindrome</button>

  <div id="result"></div>

  <script>
   function getNum() {
    return parseInt(document.getElementById("num").value);
   }

   function factorial() {
    let n = getNum(), fact = 1;
    for (let i = 1; i <= n; i++) fact *= i;
    show(`Factorial: ${fact}`);
   }

   function fibonacci() {
    let n = getNum(), a = 0, b = 1, series = [0];
    while (b <= n) [series[series.length] = b, [a, b] = [b, a + b]];
    show(`Fibonacci: ${series.join(', ')}`);
   }

   function prime() {
```

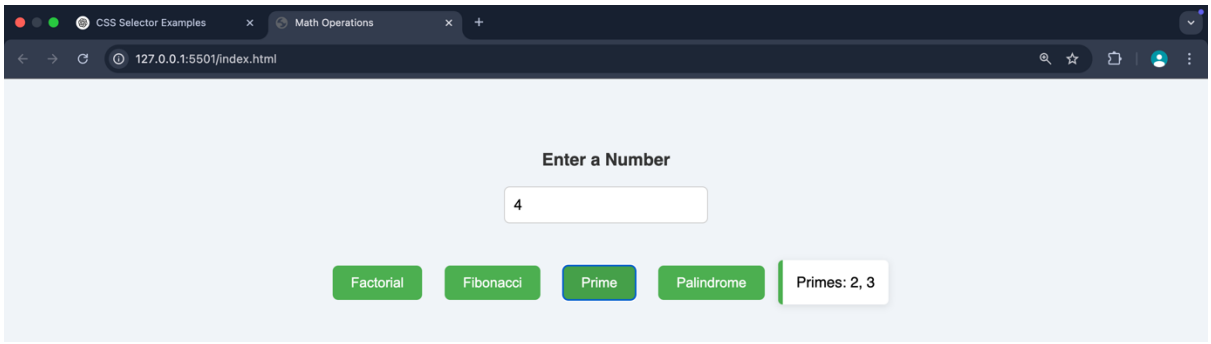
```
  let n = getNum(), primes = [];
  for (let i = 2; i <= n; i++) {
    let isP = true;
    for (let j = 2; j * j <= i; j++) if (i % j === 0) { isP = false; break; }
    if (isP) primes.push(i);
  }
  show(`Primes: ${primes.join(', ')}`);
}

function palindrome() {
  let str = getNum().toString();
  show(`${str === str.split('').reverse().join('') ? 'It is' : 'Not'} a Palindrome`);
}

function show(msg) {
  document.getElementById("result").innerHTML = msg;
}
</script>

</body>
</html>
```

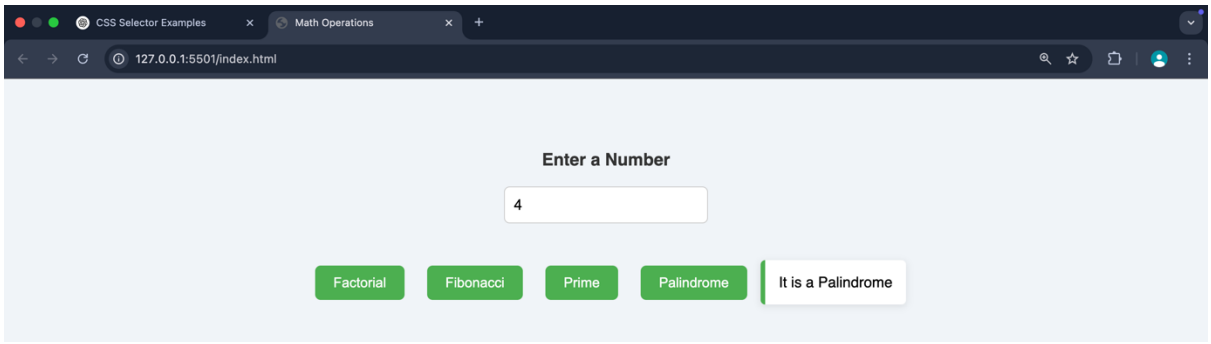
OUTPUT:




CSS Selector Examples
Math Operations
127.0.0.1:5501/index.html
Enter a Number
4
Factorial
Fibonacci
Prime
Palindrome
Factorial: 24


CSS Selector Examples
Math Operations
127.0.0.1:5501/index.html
Enter a Number
4
Factorial
Fibonacci
Prime
Palindrome
Fibonacci: 0, 1, 1, 2, 3


CSS Selector Examples
Math Operations
127.0.0.1:5501/index.html
Enter a Number
4
Factorial
Fibonacci
Prime
Palindrome
Primes: 2, 3


CSS Selector Examples
Math Operations
127.0.0.1:5501/index.html
Enter a Number
4
Factorial
Fibonacci
Prime
Palindrome
It is a Palindrome

B. Write a program to validate the following fields in a registration page

i. Name (start with alphabet and followed by alphanumeric and the length should not be less than 6 characters)

ii. Mobile(onlynumbersandlength10digits)

iii. E-mail (should contain form at like xxxxxxx@xxxxxx.xxx)

Introduction

Validating input fields in a registration page is an essential step to ensure the integrity of user-provided data. Proper validation not only guarantees compliance with specific data formats but also enhances user experience by reducing errors and preventing invalid entries. In this example, we validate the following fields:

1. **Name**: The name must start with an alphabet, may contain alphanumeric characters, and must have a minimum length of 6 characters.
2. **Mobile**: The mobile number should consist of only numerical digits and have exactly 10 characters.
3. **E-mail**: The email address must follow the standard format (e.g., xxxxxxx@xxxxxx.xxx).

The validation is achieved through JavaScript using regular expressions (Regex), which enable powerful and precise pattern matching. Below is the complete HTML and JavaScript implementation.

PROGRAMME:

```
<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <title>Registration Validation</title>

 <style>

  body { font-family: Arial; padding: 30px; background: #f5f5f5; }

  form { background: #fff; padding: 20px; max-width: 400px; margin: auto; border-radius: 8px; box-shadow: 0 0 10px #ccc; }

  input { display: block; margin: 10px 0; padding: 8px; width: 100%; box-sizing: border-box; }

  button { padding: 10px 20px; background: #4CAF50; color: white; border: none; border-radius: 4px; cursor: pointer; }
```

```
    #msg { margin-top: 10px; color: red; }
  </style>
</head>
<body>

<form onsubmit="return validateForm()">
  <h3>Register</h3>
  <input type="text" id="name" placeholder="Enter Name" required>
  <input type="text" id="mobile" placeholder="Enter Mobile Number" required>
  <input type="email" id="email" placeholder="Enter Email" required>
  <button type="submit">Submit</button>
  <div id="msg"></div>
</form>

<script>
  function validateForm() {
    const name = document.getElementById('name').value;
    const mobile = document.getElementById('mobile').value;
    const email = document.getElementById('email').value;

    const namePattern = /^[A-Za-z][A-Za-z0-9]{5,}$/;
    const mobilePattern = /^[0-9]{10}$/;
    const emailPattern = /^[\w.-]+@[a-zA-Z\d.-]+\.[a-zA-Z]{2,}$/;

    if (!namePattern.test(name)) {
      return showMsg("Invalid Name: Must start with a letter and be at least 6 characters.");
    }
    if (!mobilePattern.test(mobile)) {
      return showMsg("Invalid Mobile: Must be exactly 10 digits.");
```

```
  }
  if (!emailPattern.test(email)) {
    return showMsg("Invalid Email: Must follow format xxxxx@xxxxx.xxx");
  }

  alert("Registration Successful!");
  return true;
}

function showMsg(msg) {
  document.getElementById('msg').textContent = msg;
  return false;
}
</script>

</body>
</html>
```

OUTPUT: