

EXERCISE-1

1. HTML:

HTML stands for **Hyper Text Markup Language**. It is the basic language used to create the structure of web pages. It organizes and displays text, images, videos, and links on websites.

Use HTML:

- To **design the layout** of a webpage.
- To **display content** like headings, paragraphs, and images.
- To add **links** and create navigation between web pages.
- To work with other technologies like CSS (for design) and JavaScript (for interactivity).

Who Developed HTML and Why?

- **Developer:** Tim Berners-Lee in 1991.
- **Purpose:** He created HTML to make it easy for researchers to share and connect documents over the internet.

Advantages of HTML:

1. **Easy to learn:** Simple and beginner-friendly.
2. **Free:** No cost to use or create HTML documents.
3. **Works everywhere:** Supported by all web browsers and devices.
4. **Customizable:** Can be combined with CSS and JavaScript for advanced features.
5. **SEO-friendly:** Helps improve website visibility on search engines.

Applications of HTML:

- **Creating websites:** Blogs, business sites, and e-commerce platforms.
- **Designing forms:** For collecting user data.
- **Email templates:** For marketing emails.
- **Web applications:** Works with other tools to build interactive sites.

HTML Boilerplate (HTML structure):

The **boilerplate** is the basic structure of an HTML document that serves as a starting point for creating web pages.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Webpage</title>
  </head>
  <body>
    <h1>Welcome to My Website</h1>
  </body>
</html>
```

Description:

- `<!DOCTYPE html>`: Declares the document as an HTML5 file.
- `<html>`: The root element that contains all the content of the webpage.
- `<head>`: Contains meta-information like the title, styles, and links to external files.
- `<title>`: Sets the title of the webpage (displayed on the browser tab).
- `<body>`: Contains the visible content of the webpage, like text, images, and links.
- `<h1>`: A heading tag to display large, bold text.

LISTS, LINKS AND IMAGES:

1. HTML Lists:

- The List is a way to group related pieces of information so that they are easy to read and understand.
- `` tag is used to represent the list. Data must be enclosed between ` `.
- The default behaviour of `` tag is Unordered list type (bullets, circles, disc etc)
- For example, Shopping list, Todo list, etc.

There are mainly two types of Lists available in HTML.

1. Ordered List
2. Unordered List
3. Nested Lists
4. Definition List

1. Order list():

An ordered list numbers the items sequentially. That is

1. 1, 2, 3, 4
2. i, ii, iii, iv
3. A, B, C, D
4. A, b, c, d

It is represented by using `.... `. Default values are 1, 2, 3, ...

Example:

Code:

```
<ol>
  <li>HTML</li>
  <li>CSS</li>
  <li>JavaScript</li>
</ol>
```

Output:

1. HTML
2. CSS
3. JavaScript

2. Unordered List ():

- Displays items with bullets, where the order does not matter.
- The unordered items are bullets, circle, square, disc, etc.
- Images also used instead of bullets.
- The default type of unordered list is bullet

Example:

Code:

```
<ul>
    <li>Apple</li>
    <li>Banana</li>
    <li>Cherry</li>
</ul>
```

Output:

- Apple
- Banana
- Cherry

3. Nested Lists:

A nested list is a list inside another list.

Example:

Code: Ordered List Inside an Unordered List

```
<ul>
    <li>Fruits
        <ol>
            <li>Apple</li>
            <li>Banana</li>
        </ol>
    </li>
    <li>Vegetables
        <ol>
            <li>Carrot</li>
            <li>Spinach</li>
        </ol>
    </li>
</ul>
```

Output:

- Fruits
 1. Apple
 2. Banana
- Vegetables
 1. Carrot
 2. Spinach

4. Definition List (<dl>):

- A definition list displays terms and their descriptions.
- There are two key terms in Definition list.

They are:

- **<dt>**: Stands for **Definition Term**. It is used to specify the term being defined.
- **<dd>**: Stands for **Definition Description**. It provides the description or explanation of the term.

Example:

Code:

```
<dl>
<dt>HTML</dt>
```

```

<dd>Hyper Text Markup Language, used to create web pages.</dd>
<dt>CSS</dt>
</dl>
<dd>Cascading Style Sheets, used to style web pages.</dd>

```

Output:

- **HTML:** Hyper Text Markup Language, used to create web pages.
- **CSS:** Cascading Style Sheets, used to style web pages.

1a) 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.

SOURCE CODE: (list.html)

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>HTML Lists Example</title>
</head>
<body>
  <h1>HTML Lists Example</h1>

  <!-- Ordered List -->
  <h2>Ordered List</h2>
  <ol>
    <li>Step 1: Gather ingredients</li>
    <li>Step 2: Mix ingredients</li>
    <li>Step 3: Bake</li>
  </ol>

  <!-- Unordered List -->
  <h2>Unordered List</h2>
  <ul>
    <li>Apple</li>
    <li>Banana</li>
    <li>Cherry</li>
  </ul>

  <!-- Nested Lists -->
  <h2>Nested List</h2>
  <ul>
    <li>Fruits
      <ul>
        <li>Apple</li>
        <li>Banana</li>
      </ul>
    </li>
  </ul>

```

```

<li>Vegetables
  <ul>
    <li>Carrot</li>
    <li>Spinach</li>
  </ul>
</li>
</ul>
<!-- Definition List -->
<h2>Definition List</h2>
<dl>
  <dt>HTML</dt>
  <dd>Hyper Text Markup Language, used to structure web pages.</dd>
  <dt>CSS</dt>
  <dd>Cascading Style Sheets, used for styling web pages.</dd>
  <dt>JavaScript</dt>
  <dd>A programming language used to make web pages interactive.</dd>
</dl>
</body>
</html>

```

OUTPUT:

Ordered List

1. Step 1: Gather ingredients
2. Step 2: Mix ingredients
3. Step 3: Bake

Unordered List

- Apple
- Banana
- Cherry

Nested List

- Fruits
 - Apple
 - Banana
- Vegetables
 - Carrot
 - Spinach

Definition List

HTML

Hyper Text Markup Language, used to structure web pages.

CSS

Cascading Style Sheets, used for styling web pages.

JavaScript

A programming language used to make web pages interactive.

Hyperlinks in HTML:

Hyperlinks are used in HTML to link one page to another. They allow users to navigate between different web pages or sections within the same page.

Hyperlinks are created using the <a> (anchor) tag, with attributes like “href” and “target” to control the link's behaviour.

Attributes of the <a> Tag

1. **href** : Specifies the URL of the page the link points to.
 - Example: Visit Example
2. **target**: Specifies where to open the linked document. Common values are:
 - _self (default): Opens the link in the same tab.
 - _blank: Opens the link in a new tab.
 - _parent: Opens the link in the parent frame (used in framesets).
 - _top: Opens the link in the full browser window, removing frames.

Examples:

Basic Hyperlink:

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

Clicking this link redirects the user to Google's homepage in the same tab.

Open in a New Tab:

```
<a href="https://www.wikipedia.org" target="_blank">Visit Wikipedia</a>
```

Opens Wikipedia in a **new tab**.

Internal Linking:

```
<a href="#section2">Jump to Section 2</a>
```

The href="#section2" links to an element on the same page with id="section2".

When clicked, the browser scrolls to the specific section.

1b) Write a HTML program, to explain the working of hyperlinks using tag and href, target Attributes.

SOURCE CODE: (link.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Hyperlinks Example</title>
</head>
<body>
  <h1>Working with Hyperlinks</h1>

  <!-- Basic Hyperlink -->
  <p><a href="https://www.google.com">Visit Google</a></p>

  <!-- Open Link in a New Tab -->
  <p><a href="https://www.wikipedia.org" target="_blank">Visit Wikipedia in a New Tab</a></p>
  <!-- Link to Another Section in the Same Page -->
  <p><a href="#section2">Jump to Section 2</a></p>
  <hr>
  <!-- Section 2 -->
```

```
<h2 id="section2">Section 2: Internal Linking</h2>
<p>This section demonstrates linking to a part of the same page.</p>
```

```
</body>
```

```
</html>
```

OUTPUT:

Working with Hyperlinks

[Visit Google](#)

[Visit Wikipedia in a New Tab](#)

[Jump to Section 2](#)

Section 2: Internal Linking

This section demonstrates linking to a part of the same page.

Image:

- The tag in HTML is used to embed images into a web page.
- It is a self-closing tag or void tag , meaning it does not require a closing tag.
- Images enhance the visual appeal and usability of a website.

Syntax:

```

```

Example:

```

```

1c) 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.

SOURCE CODE: (Friend1.html)

```
<html>
<head>
<title>friend1</title>
</head>
<body>
<h1>Friend 1</h1>
<ul>
<li>Name: Jhon</li>
<li>Age: 20</li>
<li>Address: vijayawada</li>
</ul>
</body>
</html>
```

SOURCE CODE: (Friend2.html)

```
<html>
<head>
<title>friend1</title>
</head>
<body>
<h1>Friend 2</h1>
<ul>
<li>Name: Riya</li>
<li>Age: 20</li>
<li>Address: vizag</li>
</ul>
</body>
</html>
```

SOURCE CODE: (ImagePage.html)

```
<html>
<head>
<title>images</title>
</head>
<body>
<h1>Images</h1>
<a href="Friend1.html"> </a>

<a href="Friend2.html"> </a>
</body>
</html>
```

OUTPUT:

Images



Friend 1

- Name: Jhon
- Age: 20
- Address: vijayawada

1d) 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.

Explanation:

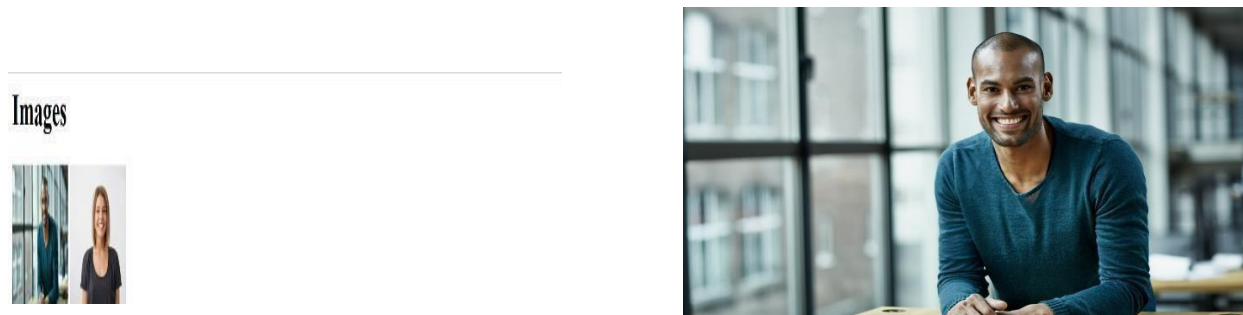
1. Create two thumbnails with height=100px and width=100px.
2. When click on the particular thumbnail, the image should be full size.

SOURCE CODE: (ThumbnailImage.html)

```
<html>
<head>
<title>images</title>
</head>
<body>
<h1>Images</h1>
<a href="https://images.inc.com/uploaded_files/image/1920x1080/getty_481292845_77896.jpg"
target="_blank">
  
</a>

<a
href="https://cdn2.psychologytoday.com/assets/styles/manual_crop_1_1_1200x1200/public/field_blog_entr
y_images/2018-09/shutterstock_648907024.jpg?itok=1-9sfjwH" target="_blank">
</a>
</body>
</html>
```

OUTPUT:



EXERCISE-2

2. HTML Tables, Forms and Frames

Table:

The <table> tag in HTML is used to create a table structure on a webpage. It allows you to display data in a grid format, with rows and columns. A table consists of several key elements that help organize and structure the data.

1. Basic Structure of a Table:

```
<table>

  <!-- Table content goes here -->

</table>
```

2. Key Tags Used Inside <table>:

- **<tr> (Table Row):** The <tr> tag is used to define a row in the table. Each row contains a set of cells, which can either be header cells or data cells.

Syntax:

```
<tr>

  <!-- Row content goes here -->

</tr>
```

- **<th> (Table Header):**
The <th> tag is used to define header cells in the table. The content inside a <th> is typically bold and centered by default. It helps label the columns and rows of the table.

Syntax:

```
<th>Header </th>
```

- **<td> (Table Data):**
The <td> tag defines a data cell in the table. Each <td> tag represents a piece of data in a particular row and column of the table.

Syntax:

```
<td>Data 1</td>
```

Example:

```
<table>

  <tr>

    <th>First Name</th>
    <th>Last Name</th>
    <th>Age</th>
```

```

</tr>
<tr>
  <td>John</td>
  <td>Doe</td>
  <td>28</td>
</tr>
<tr>
  <td>Jane</td>
  <td>Smith</td>
  <td>34</td>
</tr>
</table>

```

3. **Additional Table Tags:**

<caption>:

The <caption> tag is used to add a title or caption to the table. It appears above the table.

Example:

```
<caption>Employee Information</caption>
```

<col>:

The <col> tag defines the properties of one or more columns, such as setting the width of a column. It is used within the <colgroup> element.

Example:

```

<colgroup>
  <col style="background-color:yellow">
  <col style="background-color:lightgreen">
</colgroup>

```

<thead>, <tbody>, and <tfoot>:

These tags are used to group sections of the table.

- <thead>: Contains the header content.
- <tbody>: Contains the body/content of the table.
- <tfoot>: Contains footer rows, often used for summary or totals.

Example:

```

<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
    </tr>

```

```

</thead>
<tbody>
  <tr>
    <td>John</td>
    <td>30</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>25</td>
  </tr>
</tbody>
<tfoot>
  <tr>
    <td colspan="2">Total</td>
  </tr>
</tfoot>
</table>

```

2a) Write a HTML program, to explain the working of tables. (use tags: <th>,<tr>,<td>)

SOURCE CODE: (table.html)

```

<html>
<head></head>
<body>
<table>
  <tr>
    <th>First Name</th>
    <th>Last Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>28</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>Smith</td>
    <td>34</td>
  </tr>
</table>
</body>
</html>

```

OUTPUT:

First Name	Last Name	Age
John	Doe	28
Jane	Smith	34

Table Attributes:

1. border Attribute

The border attribute is used to specify the thickness of the table's border. It can be applied to the <table> tag. When you set the border attribute, it defines the width of the border for the table as well as the individual cells.

Example:

```
<table border="1">
  <tr>
    <th>First Name</th>
    <th>Last Name</th>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>Smith</td>
  </tr>
</table>
```

2. rowspan Attribute

The rowspan attribute is used in the <td> or <th> tag to make a cell span across multiple rows. It allows a cell to extend vertically over multiple rows.

Example:

```
<table border="1">
  <tr>
    <th rowspan="2">Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>John</td>
    <td>28</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>34</td>
  </tr>
</table>
```

3. colspan Attribute

The colspan attribute is used in the <td> or <th> tag to make a cell span across multiple columns. It allows a cell to extend horizontally over multiple columns.

Example:

```
<table border="1">
  <tr>
```

```

    <th colspan="2">Personal Information</th>
</tr>
<tr>
    <td>Name</td>
    <td>John</td>
</tr>
<tr>
    <td>Age</td>
    <td>28</td>
</tr>
</table>

```

2b) Write a HTML program, to explain the working of tables. (use tags: <th>,<tr>,<td> and attributes: border, rowspan, colspan)

SOURCE CODE: (Table.html)

```

<!DOCTYPE html>
<html>
<head>
    <title>Table with Attributes</title>
</head>
<body>
    <h2>HTML Table with Border, Rowspan, and Colspan</h2>
    <table border="1">
        <tr>
            <th rowspan="2">Name</th>
            <th colspan="2">Contact</th>
        </tr>
        <tr>
            <th>Email</th>
            <th>Phone</th>
        </tr>
        <tr>
            <td>John</td>
            <td>john@example.com</td>
            <td>123-456-7890</td>
        </tr>
        <tr>
            <td>Jane</td>
            <td>jane@example.com</td>
            <td>987-654-3210</td>
        </tr>
    </table>
</body>
</html>

```

OUTPUT:

HTML Table with Border, Rowspan, and Colspan

Name	Contact	
	Email	Phone
John	john@example.com	123-456-7890
Jane	jane@example.com	987-654-3210

2c) Write a HTML program, to explain the working of tables by preparing a timetable. (Note: Use tag to set the caption to the table & also use cell spacing, cell padding, border, rowspan, colspan etc.).

SOURCE CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Time Table</title>
</head>
<body>
  <table border="black">
    <caption> SECTION-D TIME TABLE</caption>
    <div class="class-teacher">Class Teacher: K B VARA PRASAD</div>
    <tr>
      <th></th>
      <th>1</th>
      <th>2</th>
      <th>3</th>
      <th>4</th>
      <th rowspan="8">L<br>U<br>N<br>C<br>H
    </th>
      <th>5</th>
      <th>6</th>
      <th>7</th>
    </tr>
    <tr>
      <th>DAY</th>
      <td>9:30 to 10:20</td>
      <td>10:20 to 11:10</td>
      <td>11:30 to 12:20</td>
      <td>12:20 to 1:10</td>
      <td>2:00 to 2:50</td>
      <td>2:50 to 3:40</td>
      <td>3:40 to 4:30</td>
    </tr>
    <tr>
      <th>MON</th>
```

```

        <td style="text-align: center; vertical-align: middle;">DBMS</td>
        <td style="text-align: center; vertical-align: middle;">PS</td>
        <td style="text-align: center; vertical-align: middle;">OS</td>
        <td style="text-align: center; vertical-align: middle;">SE(T)</td>
        <td style="text-align: center; vertical-align: middle;" colspan="2">***** DT&I*****</td>
        <td style="text-align: center; vertical-align: middle;">LIBRARY</td>
</tr>
<tr>
    <th>TUE</th>
        <td style="text-align: center; vertical-align: middle;">SE</td>
        <td style="text-align: center; vertical-align: middle;">DBMS</td>
        <td style="text-align: center; vertical-align: middle;">PS</td>
        <td style="text-align: center; vertical-align: middle;">OS(T)</td>
        <td style="text-align: center; vertical-align: middle;" colspan="3">****DBMS LAB****</td>
</tr>
<tr>
    <th>WED</th>
        <td style="text-align: center; vertical-align: middle;">OS</td>
        <td style="text-align: center; vertical-align: middle;">SE</td>
        <td style="text-align: center; vertical-align: middle;">DBMS</td>
        <td style="text-align: center; vertical-align: middle;">MEFA</td>

        <td style="text-align: center; vertical-align: middle;">PS(T)</td>
        <td style="text-align: center; vertical-align: middle;" colspan="2">VALUE ADDED
        COURSE</td>
</tr>
<tr>
    <th>THU</th>
        <td style="text-align: center; vertical-align: middle;">PS</td>
        <td style="text-align: center; vertical-align: middle;" colspan="3">****OS LAB****</td>
        <td style="text-align: center; vertical-align: middle;">SE</td>
        <td style="text-align: center; vertical-align: middle;">MEFA(T)</td>
        <td style="text-align: center; vertical-align: middle;">COUNSELLING</td>
</tr>
<tr>
    <th>FRI</th>
        <td style="text-align: center; vertical-align: middle;">SE</td>
        <td style="text-align: center; vertical-align: middle;">DBMS</td>
        <td style="text-align: center; vertical-align: middle;">MEFA</td>
        <td style="text-align: center; vertical-align: middle;">OS</td>
        <td style="text-align: center; vertical-align: middle;" colspan="2">****FSD-1 LAB****</td>
        <td style="text-align: center; vertical-align: middle;">REMEDIAL</td>

</tr>
<tr>
    <th>SAT</th>
        <td style="text-align: center; vertical-align: middle;">OS</td>
        <td style="text-align: center; vertical-align: middle;">DBMS(T)</td>
        <td style="text-align: center; vertical-align: middle;">MEFA</td>
<td style="text-align: center; vertical-align: middle;">PS</td>

```



```

<td style="text-align: center; vertical-align: middle;" colspan="3">*****</td>
</tr>
</table>
</body>
</html>

```

OUTPUT:

Class Teacher: K B VARA PRASAD

CSE SEC-D TIME TABLE

	1	2	3	4	L U N C H	5	6	7
DAY	9:30 to 10:20	10:20 to 11:10	11:30 to 12:20	12:20 to 1:10		2:00 to 2:50	2:50 to 3:40	3:40 to 4:30
MON	DBMS	PS	OS	SE(T)		***** DT&I*****		LIBRARY
TUE	SE	DBMS	PS	OS(T)		****DBMS LAB****		
WED	OS	SE	DBMS	MEFA		PS(T)	VALUE ADDED COURSE	
THU	PS	****OS LAB****				SE	MEFA(T)	COUNSELLING
FRI	SE	DBMS	MEFA	OS		****FSD-1 LAB****		REMEDIAL
SAT	OS	DBMS(T)	MEFA	PS		*****		

HTML Forms:

In HTML, the <form> tag is used to create a form that collects user input.

1. <form>

The <form> tag is used to define a form. It wraps all the form elements like text fields, buttons, checkboxes, etc. Attributes:

- action: Specifies the **URL** to which the form data will be sent when the form is submitted. (In this case, it's set to # as a placeholder, meaning the form data won't actually be submitted anywhere.)
- method: Specifies the **HTTP** method to be used when submitting the form. The two common methods are:
 - **GET**: Sends the form data as part of the **URL** (visible in the address bar).
 - **POST**: Sends the form data as part of the **HTTP** request body (not visible in the address bar).

Example:

```
<form action="#" method="POST">
```

2. <input>

The <input> tag is the most versatile form element used to create various input fields such as text fields, password fields, radio buttons, checkboxes, etc.

- type: Specifies the type of input field (e.g., text, password, number, date, radio, checkbox, etc.).
- name: The name attribute is used to reference the form data after the form is submitted.
- id: The id attribute is used to associate the input with a <label> element (for better accessibility).
- required: Makes the field mandatory (user must fill it out before submitting).

Example:

```

<input type="text" id="fname" name="fullname" required>
<input type="password" id="password" name="password" required>
<input type="number" id="phone" name="phone" required>

```

```
<input type="date" id="dob" name="dob" required>
```

3. <label>

The **<label>** tag is used to define a label for an **<input>** element. It improves accessibility by linking the text to the corresponding input element.

- **for:** The for attribute associates the label with an input field by referring to the input's id.

Example:

```
<label for="fname">Full Name:</label>
```

4. <textarea>

The **<textarea>** tag is used to create a multi-line text input field, allowing the user to input a longer text (e.g., a message, comments).

- **rows:** Specifies the visible number of lines.
- **cols:** Specifies the width of the text area (in characters).
- **placeholder:** Displays placeholder text inside the text area until the user starts typing.

Example:

```
<textarea id="message" name="message" rows="4" placeholder="Enter your message..."></textarea>
```

5. <select>

The **<select>** tag is used to create a drop-down list, allowing the user to select one or more options from the list.

- **name:** The name attribute is used to identify the drop-down list when the form is submitted.

<option> elements are nested inside the **<select>** element to define each option the user can choose from.

Example:

```
<select id="country" name="country" required>  
  <option value="India">India</option>  
  <option value="USA">USA</option>  
  <option value="UK">UK</option>  
  <option value="Australia">Australia</option>  
</select>
```

6. <option>

The **<option>** tag defines an individual option within a **<select>** drop-down list. The value attribute specifies the value that will be sent to the server if the option is selected.

Example:

```
<option value="India">India</option>
```

7. <button>

The **<button>** tag is used to create buttons within a form. It can be used to submit the form, reset the form, or trigger JavaScript actions.

- **type="submit":** When this button is clicked, it submits the form.
- **type="reset":** This button resets all form fields to their initial values.

Example:

```
<button type="submit">Submit</button>
```

```
<button type="reset">Reset</button>
```

2d) 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, checkboxes, radio buttons, list boxes using & tags, and two buttons ie: submit and reset. Use tables to provide a better view).

SOURCE CODE:

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

<div class="form-container">
  <h2>Registration Form</h2>
  <form action="#" method="POST">

    <table>
      <tr>
        <td><label for="fname">Full Name:</label></td>
        <td><input type="text" id="fname" name="fullname" placeholder="e.g: abd" required></td>
      </tr>
      <tr>
        <td><label for="email">Email:</label></td>
        <td><input type="text" id="email" name="email" placeholder="e.g: abd@gmail.com" required></td>
      </tr>
      <tr>
        <td><label for="password">Password:</label></td>
        <td><input type="password" id="password" name="password" placeholder="e.g: password"
          required></td>
      </tr>
      <tr>
        <td><label for="phone">Phone Number:</label></td>
        <td><input type="number" id="phone" name="phone" placeholder="e.g: 1234567" required></td>
      </tr>
      <tr>
        <td><label for="dob">Date of Birth:</label></td>
        <td><input type="date" id="dob" name="dob" required></td>
      </tr>
      <tr>
        <td><label for="gender">Gender:</label></td>
        <td>
```

```

<input type="radio" id="male" name="gender" value="male" required><label
for="male">Male</label><br>

<input type="radio" id="female" name="gender" value="female" required><label
for="female">Female</label><br>
<input type="radio" id="other" name="gender" value="other" required><label
for="other">Other</label>
</td>
</tr>

<tr>
<td><label for="skills">Skills:</label></td>
<td>
<input type="checkbox" id="skill1" name="skills" value="Reading"><label
for="skill1">C</label><br>
<input type="checkbox" id="skill2" name="skills" value="Traveling"><label
for="skill2">PYTHON</label><br>
<input type="checkbox" id="skill3" name="skills" value="Sports"><label
for="skill3">C++</label><br>
<input type="checkbox" id="skill4" name="skills" value="Sports"><label
for="skill4">JAVA</label>
</td>
</tr>

<td><label for="branch">Branch:</label></td>
<td>
<select id="branch" name="branch" required>
<option value="CIV">CIV</option>
<option value="CSE">CSE</option>

<option value="CSE-AI&DS">CSE-AI&DS</option>
<option value="CSE-AI&ML">CSE-AI&ML</option>
<option value="CSE-CS">CSE-CS</option>
<option value="ECE">ECE</option>
<option value="EEE">EEE</option>
<option value="IT">IT</option></select>
</td>
</tr>
<tr>
<td><label for="message">Message:</label></td>
<td><textarea id="message" name="message" rows="4" placeholder="Enter your
message..."></textarea></td>
</tr>
<tr>
<td colspan="2">
<button type="submit">Submit</button>
<button type="reset">Reset</button>
</td>
</tr>

```

```

</table>
</form>
</div>
</body>
</html>

```

OUTPUT:

Registration Form

Full Name:	<input type="text" value="e.g: abd"/>
Email:	<input type="text" value="e.g: abd@gmail.com"/>
Password:	<input type="text" value="e.g: password"/>
Phone Number:	<input type="text" value="e.g: 1234567"/>
Date of Birth:	<input type="text" value="dd-mm-yyyy"/> <input type="button" value="📅"/>
Gender:	<input type="radio"/> Male <input type="radio"/> Female <input type="radio"/> Other
Skills:	<input type="checkbox"/> C <input type="checkbox"/> PYTHON <input type="checkbox"/> C++ <input type="checkbox"/> JAVA
Branch:	<input type="text" value="CIV"/> ▼
Message:	<input type="text" value="Enter your message..."/>
<input type="button" value="Submit"/> <input type="button" value="Reset"/>	

HTML Frame:

<frameset> Tag:

The <frameset> tag is used to define a layout of multiple frames on a webpage. Instead of using the regular page layout with a single body, the <frameset> tag allows you to break the page into different sections (frames) that can each display separate content.

Example:

```

<frameset cols="33%,33%,34%">
  <!-- Frame Definitions -->
</frameset>

```

cols="33%,33%,34%": The cols attribute specifies that the page will be divided into three vertical columns, where the first two columns take up 33% of the width each, and the third one takes up the remaining 34%. This way, the total width of the page is distributed across the three sections.

- 33%: First frame (Image frame).
- 33%: Second frame (Paragraph frame).
- 34%: Third frame (Hyperlink frame).

Note that frames can also be set horizontally (using the rows attribute), but in this case, we are dividing the page vertically into columns.

<frame> Tag:

The **<frame>** tag is used to define an individual frame within a **<frameset>**. Each frame can load a separate HTML file, displaying different content in the same browser window.

```
<frame src="image.html" name="image_frame">
<frame src="paragraph.html" name="paragraph_frame">
<frame src="hyperlink.html" name="link_frame">
```

2e) Write a HTML program, to explain the working of frames, such that page is to be divided into 3 parts on either direction. (Note: first frame à image, second frame à paragraph, third frame à hyperlink. And also make sure of using “no frame” attribute such that frames to be fixed).

SOURCE CODE: (frame.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Frames Example</title>
</head>
<frameset cols="33%,33%,34%" >
  <!-- First Frame: Displays an Image -->
  <frame src="image2.html" name="image_frame">

  <!-- Second Frame: Displays a Paragraph -->
  <frame src="paragraph.html" name="paragraph_frame">

  <!-- Third Frame: Displays a Hyperlink -->
  <frame src="hyperlink.html" name="link_frame">
</frameset>
</html>
```

SOURCE CODE: (image.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Image Frame</title>
</head>
<body>
<h1>Image frame</h1>
  
</body>
</html>
```


SOURCE CODE: (paragraph.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Paragraph Frame</title>
</head>
<body>
<h1>Paragraph frame</h1>
  <p>This is a paragraph displayed in the second frame. You can put any content here.</p>
</body>
</html>
```

SOURCE CODE: (hyperlink.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hyperlink Frame</title>
</head>
<body>
<h1>Hyper link frame</h1>
  <a href="https://www.example.com" target="_blank">Click here to visit Example.com</a>
</body>
</html>
```

OUTPUT:

Image frame	Paragraph frame	Hyper link frame
	This is a paragraph displayed in the second frame. You can put any content here.	Click here to visit Example.com

EXERCISE-3

HTML 5 and Cascading Style Sheets, Types of CSS:

3a. Write a HTML program, that makes use of <article>, <aside>, <figure>, <figcaption>, <footer>, <header>, <main>.<nav>,<section>,<div>, tags.

1. <header>

- **Purpose:** Represents the header of the page or a section of the page. It typically contains introductory content, such as a logo, site title, navigation menu, or a tagline.
- **Common Usage:** Used at the top of the page or within sections to group content that serves as an introduction to the page or section.

Example:

```
<header>
  <h1>My Web Page</h1>
  <p>Welcome to my webpage that uses semantic HTML tags!</p>
</header>
```

2. <nav>

- **Purpose:** Defines a navigation block in the document. It groups links together, typically used for menus or navigation bars.
- **Common Usage:** Used for creating navigation menus or linking sections within a page or across websites.

Example:

```
<nav>
  <a href="#">Home</a>
  <a href="#">About</a>
  <a href="#">Services</a>
  <a href="#">Contact</a>
</nav>
```

3. <main>

- **Purpose:** Represents the main content of the document. It contains the central content directly related to or expanding on the central topic of the page. This content should be unique to the page and not repeated on other pages.
- **Common Usage:** It's used to wrap the primary content of the page, excluding headers, footers, and navigation.

Example:

```
<main>
  <article>
    <section>
      <h2>Main Article</h2>
      <p>This is the main article of the page.</p>
    </section>
  </article>
</main>
```

4. <article>

- **Purpose:** Represents a self-contained piece of content that could stand alone and be reused or distributed separately (such as a news article, blog post, or forum entry).
- **Common Usage:** Each <article> element contains a distinct block of content, which can be independently understood or syndicated.

Example:

```
<article>
  <section>
    <h2>Main Article</h2>
    <p>This is the main article of the page.</p>
  </section>
</article>
```

5. <section>

- **Purpose:** Represents a thematic grouping of content, typically with a heading. It is used to break the page into sections of related content, like chapters or groups of content within an article.
- **Common Usage:** Used within <article> or other elements to divide the page into sections that each have a unique subheading.

Example:

```
<section>
  <h2>Another Section</h2>
  <p>This section provides additional information.</p>
</section>
```

6. <aside>

- **Purpose:** Represents content that is tangentially related to the main content, such as sidebars, pull quotes, or related links. It often provides supplementary information.
- **Common Usage:** Typically used for sidebars, advertisements, or information that is related to but not central to the page content.

Example:

```
<aside>
  <h3>Related Information</h3>
  <p>This section contains related information.</p>
</aside>
```

7. <footer>

Purpose: Represents the footer section of a page or a section. It typically contains information like the copyright, links to legal documents (e.g., privacy policy), and contact information.

Common Usage: Placed at the bottom of a page or a section to provide closing information or links.

Example:

```
<footer>
  <p>Footer Content © 2024</p>
  <div>
    <span>Privacy Policy</span> | <span>Terms of Use</span>
  </div>
</footer>
```

8. <figure>

- **Purpose:** Represents content like images, videos, or diagrams that are referenced in the document, and can include a <figcaption> for a description.
- **Common Usage:** Used to group media content along with a caption.

Example:

```
<figure>
  
  <figcaption>This is an image caption.</figcaption>
```

</figure>

9. <figcaption>

- **Purpose:** Provides a caption or description for a <figure>. It should be used to explain the content inside the <figure> tag, such as an image or video.
- **Common Usage:** Placed inside <figure> to describe its contents (like describing an image or video).

Example:

```
<figcaption>This is an image caption.</figcaption>
```

10. <div>

- **Purpose:** A general-purpose container for grouping elements and applying styles or scripting. It has no inherent meaning, but it is often used to structure content.
- **Common Usage:** Used to group together HTML elements for styling or scripting purposes (e.g., wrapping a section of the page to apply CSS).

Example:

```
<div>
```

```
<span>Privacy Policy</span> | <span>Terms of Use</span>
```

```
</div>
```

11.

- **Purpose:** A generic inline container for styling or scripting purposes. It does not create a new block of content but allows styling of specific portions of text or other inline elements.
- **Common Usage:** Used to style small portions of content inline within a paragraph or other inline elements.

Example:

```
<span>Privacy Policy</span> | <span>Terms of Use</span>
```

SOURCE CODE: (semanticTags.html)

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

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

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

```
<title>HTML Semantic Tags Example</title>
```

```
<style>
```

```
body {
```

```
font-family: Arial, sans-serif;
```

```
line-height: 1.6;
```

```
margin: 0;
```

```
padding: 0;
```

```
background-color: #f4f4f4;
```

```
}
```

```
header, footer {
```

```
background-color: #333;
```

```
color: white;
```

```
padding: 10px 0;
```

```
text-align: center;
```

```
}
```

```
nav {
```

```
background-color: #444;
```

```
padding: 10px;
```

```
text-align: center;
```

```
}
```

```

nav a {
  color: white;
  margin: 0 15px;
  text-decoration: none;
}
main {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  padding: 20px;
}
article {
  flex: 3;
  background-color: white;
  padding: 20px;
  margin-right: 10px;
}
aside {
  flex: 1;
  background-color: #f9f9f9;
  padding: 20px;
  border-left: 2px solid #ccc;
}
section {
  margin-bottom: 20px;
}
footer {
  font-size: 0.8em;
}
figure {
  display: inline-block;
  margin: 0;
  text-align: center;
}
figcaption {
  font-size: 0.9em;
  color: #555;
}
</style>

</head>
<body>
  <header>
    <h1>My Web Page</h1>
    <p>Welcome to my webpage that uses semantic HTML tags!</p>
  </header>

  <nav>
    <a href="#">Home</a>
    <a href="#">About</a>

```

```

    <a href="#">Services</a>
    <a href="#">Contact</a>
</nav>

<main>
  <article>
    <section>
      <h2>Main Article</h2>

      <p>This is the main article of the page. It contains the most important content.</p>
      <h1>Importance of flowers</h1>

    </section>

    <section>
      <h2>Another Section</h2>
      <p>This section of the article provides additional information.</p>
    </section>

    <figure>
      
      <figcaption>Flowers</figcaption>
    </figure>
  </article>

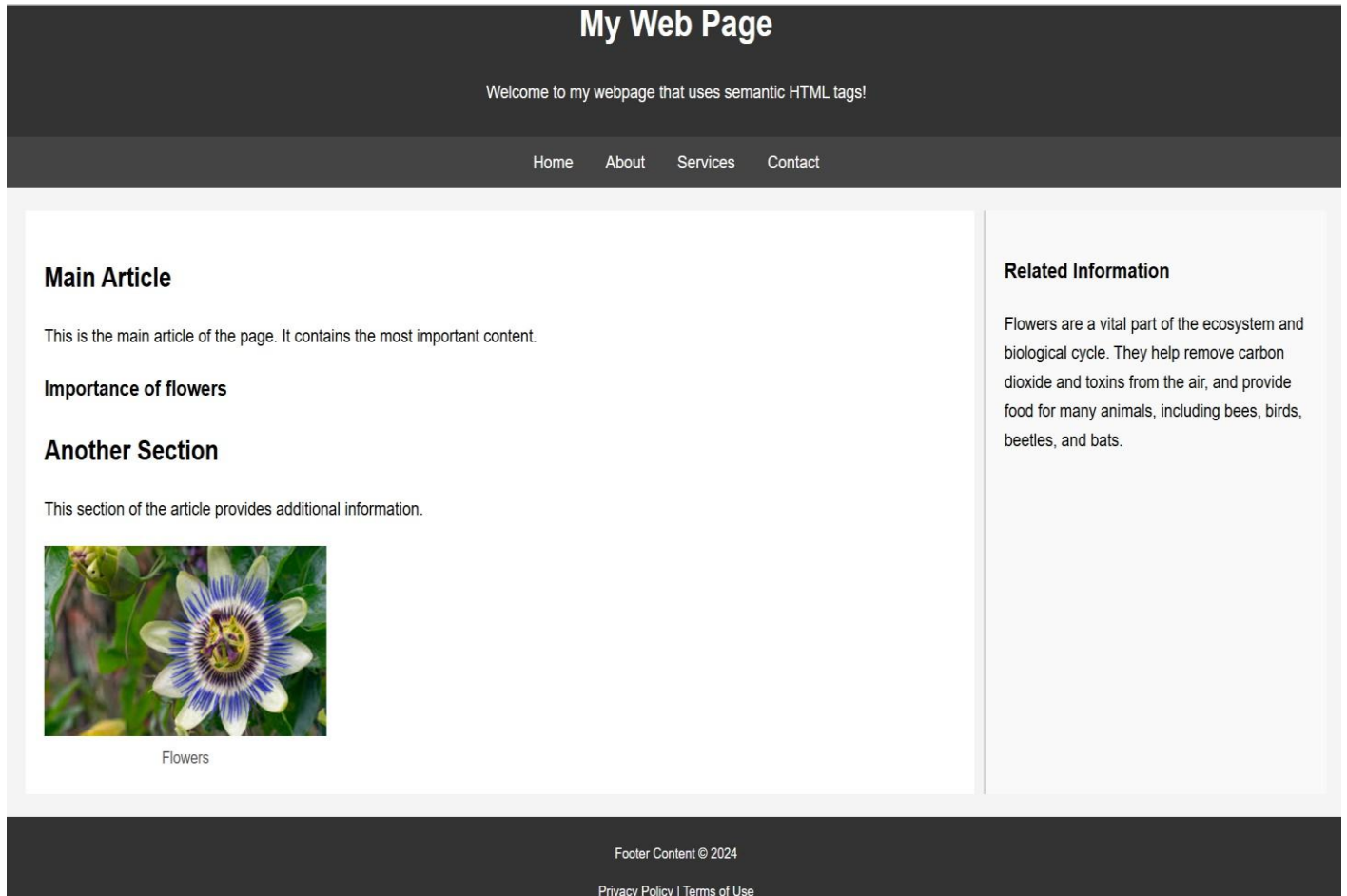
  <aside>
    <h3>Related Information</h3>
    <p>Flowers are a vital part of the ecosystem and biological cycle. They help remove carbon dioxide
and toxins from the air, and provide food for many animals, including bees, birds, beetles, and bats. </p>

  </aside>
</main>

<footer>
  <p>Footer Content © 2024</p>
  <div>
    <span>Privacy Policy</span> | <span>Terms of Use</span>
  </div>
</footer>
</body>
</html>

```

OUTPUT:



3b. Write a HTML program, to embed audio and video into HTML web page.

1. <audio>:

- The <audio> tag is used to embed an audio file in a webpage.
- The controls attribute adds audio controls (play, pause, volume, etc.) for the user to interact with.
- Inside the <audio> tag, the <source> element specifies the path to the audio file and its MIME type (in this case, MP3).
- The text "Your browser does not support the audio element" is displayed if the browser does not support the <audio> tag.

Example:

<audio controls>

<source src="audio-example.mp3" type="audio/mp3">

Your browser does not support the audio element.

</audio>

2. <video>:

- The <video> tag is used to embed a video file in a webpage.
- The controls attribute adds video controls (play, pause, volume, etc.) for user interaction.
- The width attribute sets the width of the video player (in this case, 600px).
- Inside the <video> tag, the <source> element specifies the path to the video file and its MIME type (in this case, MP4).

- The text "Your browser does not support the video element" is displayed if the browser does not support the <video> tag.

Example:

```
<video controls width="600">
  <source src="video-example.mp4" type="video/mp4">
  Your browser does not support the video element.
</video>
```

Customization:

- **Audio File:** Replace "audio-example.mp3" with the path to your desired audio file.
- **Video File:** Replace "video-example.mp4" with the path to your desired video file.
- You can also adjust the width and height of the video using the width and height attributes, depending on the layout you want.

Important Notes:

- Make sure the audio and video files are in a compatible format (e.g., MP3 for audio and MP4 for video) that is widely supported by most browsers.
- You can use multiple <source> elements within the <audio> or <video> tags to provide different formats for compatibility (e.g., WebM, Ogg).

SOURCE CODE: (Audio_Video.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Embed Audio and Video</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
      background-color: #f4f4f4;
    }
    h1 {
      text-align: center;
      color: #333;
    }
    .media-container {
      text-align: center;
      margin-top: 20px;
    }
    audio, video {
      margin-top: 20px;
      max-width: 100%;
    }
  </style>
</head>
<body>

  <h1>Embedding Audio and Video in HTML</h1>
  <div class="media-container">
    <h2>Audio Example</h2>
```

```
<audio controls>
  <source src="E:\Nannaku.mp3" type="audio/mp3">
  Your browser does not support the audio element.
</audio>

<h2>Video Example</h2>
<video controls width="600">
  <source src="E:\Amma Song.mp4 " type="video/mp4">
  Your browser does not support the video element.
</video>
</div>

</body>
</html>
```

OUTPUT:

Embedding Audio and Video in HTML

Audio Example

▶ 0:25 / 3:09 ————— 🔊 ⋮

Video Example



3c. 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).

1. Inline CSS

Definition: Inline CSS is used to apply styles directly to individual HTML elements using the style attribute within the element's tag. This method allows you to define styles for specific elements without affecting other elements on the page.

Usage:

- It is often used when you want to apply a style to a single element and do not need to reuse it elsewhere.
- It takes precedence over both internal and external styles for the specific element because it is applied directly.

2. Internal CSS

Definition: Internal CSS is used when you define styles within a <style> tag, which is placed in the <head> section of the HTML document. This method allows you to apply styles to an entire HTML page.

Usage:

- It is useful when you want to style a single HTML document and keep the styles separate from the content (but still within the same document).
- Internal CSS can be used when the styles are not needed across multiple pages, but you want to keep styles grouped together within a single file.

3. External CSS

Definition: External CSS involves writing the CSS styles in a separate .css file and linking that file to the HTML documents using the <link> tag. This is the most scalable and maintainable approach for styling websites, especially when you want to apply consistent styles across multiple pages.

Usage:

- It is ideal for large websites or projects where the same style needs to be applied across multiple HTML files.
- External CSS helps in keeping the HTML content separate from the styling, making the code cleaner and more maintainable.

Summary of Differences:

Feature	Inline CSS	Internal CSS	External CSS
Location	Directly in HTML element (via style attribute).	Inside <style> tag in the <head> section.	In a separate .css file linked using <link> tag.
Scope	Only applies to the specific element.	Applies to the entire HTML document.	Applies to all linked HTML documents.
Reusability	No reuse, each element needs inline styles.	Only reusable within the same document.	Highly reusable across multiple pages.
Maintainability	Difficult to maintain, especially for large projects.	Easier to maintain within a single page, but limited to one document.	Highly maintainable and scalable for large websites.
Performance	Can make the page heavier, especially if many elements use inline styles.	No significant performance impact, but style rules are in the HTML document.	Optimized performance with caching, but requires an extra HTTP request.
Use Case	Small tweaks or testing styles on individual elements.	Small websites or single-page projects.	Large websites with multiple pages that need to share common styles.

CSS CODE: (styles.css)

/* External CSS in "styles.css" file */

```
.highlight {  
  padding: 20px; /* Property: padding, Value: 20px */  
  border: 1px solid black; /* Property: border, Value: 1px solid black */  
  margin-top: 10px; /* Property: margin-top, Value: 10px */  
  background-color: lightgray; /* Property: background-color, Value: lightgray */  
}
```

SOURCE CODE: (code.html)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>CSS Style Types</title>

<!-- This links to an external CSS file named "styles.css" -->

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

<!-- Internal CSS (Styles written within <style> tag) -->

<style>

/* This is an internal style */

h1 {

color: #4CAF50; /* Property: color, Value: #4CAF50 */

text-align: center; /* Property: text-align, Value: center */

}

p {

font-size: 16px; /* Property: font-size, Value: 16px */

color: darkblue; /* Property: color, Value: darkblue */

}

.highlight {

background-color: yellow; /* Property: background-color, Value: yellow */

font-weight: bold; /* Property: font-weight, Value: bold */

}

</style>

</head>

<body>

<h1>This is a Heading with Internal Style</h1>

<p>This is a paragraph with internal styles.</p>

<!-- Inline CSS (styles written directly in the element) -->

<p style="color: red; font-size: 18px;">This is a paragraph with inline style.</p>

<!-- External CSS (Styles applied via external CSS file) -->

<!--

Assuming you have an external CSS file "styles.css" with styles defined for .box class.
The link tag below includes that external CSS file.

-->

```
<link rel="stylesheet" href="styles.css">
```

```
<div class="highlight">
```

This is a div with a class, styled using internal CSS (or external if styles.css includes .highlight).

```
</div>
```

```
<!-- Inline CSS on a button -->
```

```
<button style="background-color: blue; color: white; padding: 10px;">Click Me</button>
```

```
</body>
```

```
</html>
```

OUTPUT:

This is a Heading with Internal Style

This is a paragraph with internal styles.

This is a paragraph with inline style.

This is a div with a class, styled using internal CSS (or external if styles.css includes .highlight).

Click Me

EXERCISE-4

4. 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

i. Simple selector (element, id, class, group, universal) CSS

Selectors:

CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

1. **Simple selectors** (select elements based on name, id, class)
2. **Combinator selectors** (select elements based on a specific relationship between them)
3. **Pseudo-class selectors** (select elements based on a certain state)
4. **Pseudo-elements selectors** (select and style a part of an element)
5. **Attribute selectors** (select elements based on an attribute or attribute value)

1. Simple Selectors:

- **Element Selector:** Targets all elements of a specific type (tag) in the HTML.
 - Example: `p { color: blue; }` targets all `<p>` elements.
- **ID Selector:** Targets an element with a specific id.
 - Example: `#header { font-size: 24px; }` targets the element with `id="header"`.
- **Class Selector:** Targets all elements with a specific class.
 - Example: `.highlight { background-color: yellow; }` targets all elements with `class="highlight"`.
- **Group Selector:** Applies the same styles to multiple elements.
 - Example: `h1, h2, h3 { color: red; }` targets `<h1>`, `<h2>`, and `<h3>` elements.
- **Universal Selector:** Targets all elements on the page.
 - Example: `* { font-family: Arial; }` applies a font to every element.

SOURCE CODE: (simpleselector.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>All Simple CSS Selectors</title>
<style>
  /* Element Selector */
  p {
    color: blue;
  }
```

```

/* ID Selector */
#header {
    color: green;
    font-size: 24px;
}
/* Class Selector */
.highlight {
    background-color: yellow;
    font-weight: bold;
}
/* Group Selector */
h1, h2, h3 {
    color: red;
}

/* Universal Selector */
* {
    font-family: Arial, sans-serif;
}
</style>
</head>
<body>
<h1>This is a red heading (Group Selector)</h1>
<h2>This is another red heading (Group Selector)</h2>
<h3>This is yet another red heading (Group Selector)</h3>

<p>This paragraph will be blue (Element Selector).</p>
<p class="highlight">This paragraph has a yellow background (Class Selector).</p>

<div id="header">This is a green heading with ID selector.</div>

<p class="highlight">This paragraph also has a yellow background (Class Selector).</p>

<p>All elements in the document use Arial font (Universal Selector).</p>
</body>
</html>

```

OUTPUT:

This is a red heading (Group Selector)

This is another red heading (Group Selector)

This is yet another red heading (Group Selector)

This paragraph will be blue (Element Selector).

This paragraph has a yellow background (Class Selector).

This is a green heading with ID selector.

This paragraph also has a yellow background (Class Selector).

All elements in the document use Arial font (Universal Selector).

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

The Combinator Selectors in CSS are used to select elements based on their relationship with other elements. They allow you to target elements that are positioned relative to others in the HTML structure. There are four main types of combinator selectors:

1. Descendant Selector (space)

- This selector targets an element that is a descendant (child, grandchild, etc.) of another element.

2. Child Selector (>)

- This selector targets only direct children of an element (ignoring deeper descendants).

3. Adjacent Sibling Selector (+)

- This selector targets an element that is immediately adjacent to (next to) another element.

4. General Sibling Selector (~)

- This selector targets all sibling elements that come after a specified element.

SOURCE CODE: (combinatorselector.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Combinator Selectors</title>
  <style>
    /* Descendant Selector */
    div p {
      color: blue;
    }
    /* Child Selector */
    div > p {
      color: green;
    }
    /* Adjacent Sibling Selector */
    h2 + p {
      color: red;
    }
    /* General Sibling Selector */
    h2 ~ p {
      color: purple;
    }
  </style>
</head>
<body>
```

```

<div>
  <p>This paragraph inside a div will be blue (descendant selector).</p>
  <p>This paragraph inside a div will be green (child selector).</p>
</div>

<h2>This is a heading 2</h2>
<p>This paragraph is immediately after the h2 heading and will be red (adjacent sibling selector).</p>
<p>This paragraph is also after the h2 heading but will be purple (general sibling selector).</p>
<p>This paragraph will also be purple (general sibling selector).</p>
</body>
</html>

```

OUTPUT:

This paragraph inside a div will be blue (descendant selector).

This paragraph inside a div will be green (child selector).

This is a heading 2

This paragraph is immediately after the h2 heading and will be red (adjacent sibling selector).

This paragraph is also after the h2 heading but will be purple (general sibling selector).

This paragraph will also be purple (general sibling selector).

iii. Pseudo-Class Selector in CSS

A pseudo-class selector is used to define the special state of an element. It targets elements in specific states like when an element is being hovered over, when it is the first child, or when a link has been clicked, among other scenarios.

There are various pseudo-classes, such as `:hover`, `:first-child`, `:last-child`, `:nth-child()`, etc.

Common Pseudo-Class Selectors:

1. **`:hover`** — Targets an element when it is being hovered over by the mouse pointer.
2. **`:first-child`** — Selects the first child of a specific parent.
3. **`:last-child`** — Selects the last child of a specific parent.
4. **`:nth-child()`** — Selects elements based on their position within a parent.
5. **`:visited`** — Targets links that have been visited.
6. **`:active`** — Targets an element while it is being clicked.
7. **`:focus`** — Targets an element when it has focus, such as an input field.

SOURCE CODE: (PseudoClassSelector):

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Pseudo-class Selectors Demo</title>

```

```

<style>
  button:hover
  {
    background-color: red;
  }
  button:active
  {
    background-color: green;
  }
  input:focus
  {
    background-color: red;
  }
  ul li:first-child
  {
    font-weight: bold;
  }
  ul li:last-child
  {
    color: purple;
  }
  ul li:nth-child(2)
  {
    color: orange;
  }
  a:visited
  {
    color: rgb(252, 71, 5);
  }
  p:not(.highlight)
  {
    color: gray;
  }
</style>
</head>
<body>
  <h1>Pseudo-class Selectors Demo</h1>
  <button> click</button>
  <a href="https://www.google.com"> google</a>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
    <li>Item 4</li>
  </ul>
  <input type="text" placeholder="click on me change colour">
  <p class="highlight">This paragraph is highlighted.</p>
  <p>This paragraph is not highlighted.</p>
</body>
</html>

```

OUTPUT:

Pseudo-class Selectors Demo

google

- **Item 1**
- Item 2
- Item 3
- Item 4

This paragraph is highlighted.

This paragraph is not highlighted.

iv. Pseudo-Element Selectors in CSS

A pseudo-element selector in CSS is used to style specific parts of an element or to create content that does not exist in the document structure. These selectors allow you to target parts of an element such as the first letter, the first line of text, or insert content before or after an element.

Common Pseudo-Element Selectors:

1. **:before** — Inserts content before an element's actual content.
2. **:after** — Inserts content after an element's actual content.
3. **:first-letter** — Styles the first letter of an element's content.
4. **:first-line** — Styles the first line of an element's content.

SOURCE CODE: (PseudoElementSelector.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Pseudo-element Selectors Demo</title>
  <style>
    h2::before {
      content: ">> ";
      color: blue;
    }
    h2::after {
      content: " <<";
      color: green;
    }
    p::first-letter {
      font-size: 2em;
      font-weight: bold;
      color: orange;
    }
    p::first-line {
      font-style: italic;
```



```

color: purple;
}
::selection {
background-color: yellow;
color: red;
}
p {
font-family: Arial, sans-serif;
line-height: 1.6;
}
</style>
</head>
<body>
<h2>CSS Pseudo-elements</h2>
<p>
CSS pseudo-element selectors allow you to style parts of an element or add content before or
after an element. For example, you can style the first letter of a paragraph or insert extra content
using `::before` and `::after`.
</p>
<p>
This is another example paragraph. Notice how the first letter and first line have different
styles.<br>
this is second line
</p>
</body>
</html>

```

OUTPUT:

>> CSS Pseudo-elements <<

CSS pseudo-element selectors allow you to style parts of an element or add content before or after an element. For example, you can style the first letter of a paragraph or insert extra content using `::before` and `::after`.

This is another example paragraph. Notice how the first letter and first line have different styles.
this is second line

v. Attribute Selector in CSS:

An attribute selector in CSS is used to select elements based on the presence or value of an attribute. It allows you to target elements that have specific attributes, or that match a pattern in an attribute value. This is useful for styling elements that have certain attributes like id, class, href, src, alt, and so on.

Types of Attribute Selectors

1. **[attribute]** — Selects elements that have the specified attribute, regardless of the attribute's value.
2. **[attribute="value"]** — Selects elements with a specific attribute value.
3. **[attribute~="value"]** — Selects elements whose attribute value contains a specific word.
4. **[attribute|= "value"]** — Selects elements whose attribute value is a hyphen-separated list that starts with a specific word.

5. **[attribute^="value"]** — Selects elements whose attribute value starts with a specific prefix.
6. **[attribute\$="value"]** — Selects elements whose attribute value ends with a specific suffix.
7. **[attribute*="value"]** — Selects elements whose attribute value contains a specific substring.

SOURCE CODE: (attributeSelector.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Attribute Selectors</title>
  <style>
    /* [attribute] - selects elements with a specific attribute */
    a[href] {
      color: green;
    }

    /* [attribute="value"] - selects elements with a specific attribute value */
    a[href="https://www.example.com"] {
      color: blue;
    }

    /* [attribute~="value"] - selects elements with an attribute containing a specific word */
    p[class~="highlight"] {
      background-color: yellow;
    }

    /* [attribute|= "value"] - selects elements with an attribute value starting with specific word */
    a[href|= "https"] {
      font-weight: bold;
    }

    /* [attribute^="value"] - selects elements with an attribute value starting with specific prefix */
    img[src^="https://"] {
      border: 2px solid blue;
    }

    /* [attribute$="value"] - selects elements with an attribute value ending with specific suffix */
    img[src$=".jpg"] {
      border: 2px dashed red;
    }

    /* [attribute*="value"] - selects elements with an attribute value containing a specific substring */
    a[href*="example"] {
      text-decoration: underline;
    }
  </style>
</head>
```

```
<body>
```

```
<a href="https://www.example.com">Visit Example.com</a>
```

```
<a href="https://www.other.com">Visit Other.com</a>
```

```
<p class="highlight">This paragraph will have a yellow background because it has a class with the word 'highlight'.</p>
```

```
<p class="info highlight">This paragraph also has a yellow background because it contains 'highlight' in the class.</p>
```

```

```

```

```

```
</body>
```

```
</html>
```

OUTPUT:

[Visit Example.com](https://www.example.com) [Visit Other.com](https://www.other.com)

This paragraph will have a yellow background because it has a class with the word 'highlight'.

This paragraph also has a yellow background because it contains 'highlight' in the class.



EXERCISE-5

5. CSS with Color, Background, Font, Text and CSS Box Model

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

In CSS, colors can be referenced in multiple ways to style elements. Each method provides different levels of flexibility and readability.

1. Named Colors

CSS provides a set of predefined color names that can be used directly.

Example:

```
color: red;
```

```
color: blue;
```

```
color: lightgreen;
```

Some common named colors: red, blue, green, yellow, purple, black, white, gray, etc.

2. HEX (Hexadecimal) Notation

Colors can be specified using a six-digit or three-digit hexadecimal format.

Example:

```
color: #FF5733; /* 6-digit HEX */
```

```
color: #F53; /* 3-digit shorthand HEX */
```

```
color: #000000; /* Black */
```

```
color: #ffffff; /* White */
```

The format #RRGGBB (6-digit) and #RGB (3-digit) represent red, green, and blue values.

3. RGB (Red, Green, Blue)

Colors can be defined using the RGB function with integer values (0-255).

Example:

```
color: rgb(255, 87, 51); /* Bright orange */
```

```
color: rgb(0, 0, 0); /* Black */
```

```
color: rgb(255, 255, 255); /* White */
```

4. RGBA (Red, Green, Blue, Alpha)

Similar to RGB but includes an alpha value (0 to 1) to set transparency.

Example:

```
color: rgba(255, 87, 51, 0.5); /* Semi-transparent orange */
```

```
color: rgba(0, 0, 0, 0.75); /* 75% opacity black */
```

5. HSL (Hue, Saturation, Lightness)

HSL values describe colors based on their hue (0-360 degrees), saturation (0-100%), and lightness (0-100%).

Example:

```
color: hsl(0, 100%, 50%); /* Red */
color: hsl(240, 100%, 50%); /* Blue */
color: hsl(120, 100%, 50%); /* Green */
```

6. HSLA (Hue, Saturation, Lightness, Alpha)

HSLA extends HSL by adding an alpha channel for transparency.

Example:

```
color: hsla(0, 100%, 50%, 0.5); /* Semi-transparent red */
color: hsla(240, 100%, 50%, 0.8); /* 80% opacity blue */
```

7. CSS Custom Properties (Variables)

Custom properties allow you to define and reuse colors across the stylesheet.

Example:

```
:root {
  --primary-color: #3498db;
  --secondary-color: rgb(255, 87, 51);
}

color: var(--primary-color);
color: var(--secondary-color);
```

SOURCE CODE: (colors.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    .named-color { color: blue; } /* Named color */
    .hex-color { color: #FF5733; } /* Hexadecimal */
    .rgb-color { color: rgb(255, 0, 0); } /* RGB */
    .rgba-color { color: rgba(0, 255, 0, 0.5); } /* RGBA */
    .hsl-color { color: hsl(120, 100%, 50%); } /* HSL */
    .hsla-color { color: hsla(240, 100%, 50%, 0.5); } /* HSLA */
  </style>
</head>
<body>
  <h1>CSS Color References</h1>
  <p class="named-color">This text is blue (named color).</p>
  <p class="hex-color">This text is a bright orange (#FF5733).</p>
```

```

<p class="rgb-color">This text is red (RGB).</p>
<p class="rgba-color">This text is green with 50% transparency (RGBA).</p>
<p class="hsl-color">This text is green (HSL).</p>
<p class="hsla-color">This text is blue with 50% transparency (HSLA).</p>
</body>
</html>

```

OUTPUT:



b. 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.

CSS rules:

1. **Places a background image halfway down the page** – The image should start at 50% of the viewport height (vertically centered halfway down the page).
2. **Tilts the image horizontally** – This means flipping or rotating the image across the horizontal axis.
3. **Fixed position when scrolling** – The image should not move when the user scrolls up or down the page.

SOURCE CODE: (backgroundImage.html)

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Background Image Halfway Down & Flipped</title>
  <style>
    body {
      margin: 0;
      height: 200vh; /* Make the page longer to enable scrolling */
    }
    .background-image {

```

```

    position: fixed;
    top: 50%;
    left: 0;
    width: 100%;
    height: 100%;
    background: url('https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQ_K-RTvJAuuMbCjKjN_7cIbPx0CpQOqxz00w&s') center/cover no-repeat;
    transform: scaleX(-1);
    z-index: -1;
}
.content {
    position: relative;
    color: black;
    text-align: left;
    font-size: 24px;
    padding: 50px 20px;
}
.spacer {
    height: 150vh;
}
</style>
</head>
<body>
  <div class="content">
    <h1>Background Image Starting Halfway Down</h1>
    <p>Scroll down to see the effect.</p>
  </div>
  <div class="spacer"></div>
  <div class="content">
    <p>More content here...</p>
  </div>
  <div class="background-image"></div>
</body>
</html>

```

Output:

Background Image Starting Halfway Down

Scroll down to see the effect.



c. Write a program using the following terms related to CSS font and text:

i. font-size ii. font-weight iii. font-style iv. text-decoration v. text-transformation
vi. text-alignment.

i. font-size

The font-size property controls the **size** of the text. It can be specified using various units of measurement:

- **Pixels (px)**: A fixed size (e.g., font-size: 16px;).
- **Ems (em)**: Relative to the font size of the parent element (e.g., font-size: 2em; makes the text 2 times the size of the parent element).
- **Rems (rem)**: Relative to the font size of the root element (e.g., font-size: 1.5rem;).
- **Percent (%)**: Percentage of the parent element's font size (e.g., font-size: 120%;).
- **Viewport width (vw)**: Relative to the viewport's width (e.g., font-size: 5vw;)

Example:

```
p {  
    font-size: 18px;  
}
```

ii. font-weight

The font-weight property defines the **thickness** of the text. It can be set to several values:

- **normal**: The default weight (usually 400).
- **bold**: Makes the text bold (usually 700).
- **lighter**: Makes the text thinner than normal.
- **bolder**: Makes the text thicker than normal.
- **Numeric values**: You can specify weight using numeric values between 100 and 900 (e.g., font-weight: 300; for light, font-weight: 700; for bold).

Example:

```
h1 {  
    font-weight: bold;  
}
```

iii. font-style

The font-style property determines the **style** of the text, such as making it italic or oblique:

- **normal**: The text appears in its normal style (default).
- **italic**: Makes the text italicized.

- **oblique:** Slants the text, similar to italic, but it may not always behave the same as italic depending on the font.

Example:

```
p {  
    font-style: italic;  
}
```

iv. text-decoration

The text-decoration property is used to add decorations to text, such as underlining or strikethrough:

- **underline:** Underlines the text.
- **line-through:** Adds a line through the text (strikethrough).
- **overline:** Adds a line above the text.
- **none:** Removes any decoration (e.g., to remove the underline from links).

Example:

```
a {  
    text-decoration: underline;  
}
```

v. text-transform

The text-transform property controls the **capitalization** of text:

- **uppercase:** Converts all text to uppercase letters.
- **lowercase:** Converts all text to lowercase letters.
- **capitalize:** Capitalizes the first letter of each word.
- **none:** No transformation is applied (default)

Example:

```
h2 {  
    text-transform: uppercase;  
}
```

vi. text-align

The text-align property is used to **align** the text within its containing element:

- **left:** Aligns the text to the left (default for most browsers).
- **right:** Aligns the text to the right.
- **center:** Centers the text horizontally.
- **justify:** Stretches the text so that each line of text spans the entire width of its container, with spaces between words adjusted accordingly.

Example:

```
p {  
    text-align: center;  
}
```

SOURCE CODE: cssfont_text.html

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="UTF-8">  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
<title>CSS Font and Text Example</title>  
<style>  
body {  
    font-family: Arial, sans-serif;  
    margin: 20px;  
}  
h1 {  
    font-size: 40px;  
    font-weight: bold;  
    font-style: italic;  
    text-decoration: underline;  
    text-transform: uppercase;  
    text-align: center;  
}  
p {  
    font-size: 20px;  
    font-weight: normal;  
    font-style: normal;  
    text-decoration: none;  
    text-transform: capitalize;  
    text-align: justify;  
}  
</style>  
</head>  
<body>  
<h1>CSS Font and Text Properties</h1>  
<p>  
The text is styled using font size, font weight, font style, text decoration, text transformation,  
and text alignment.  
</p>  
</body>  
</html>
```

OUTPUT:**CSS FONT AND TEXT PROPERTIES**

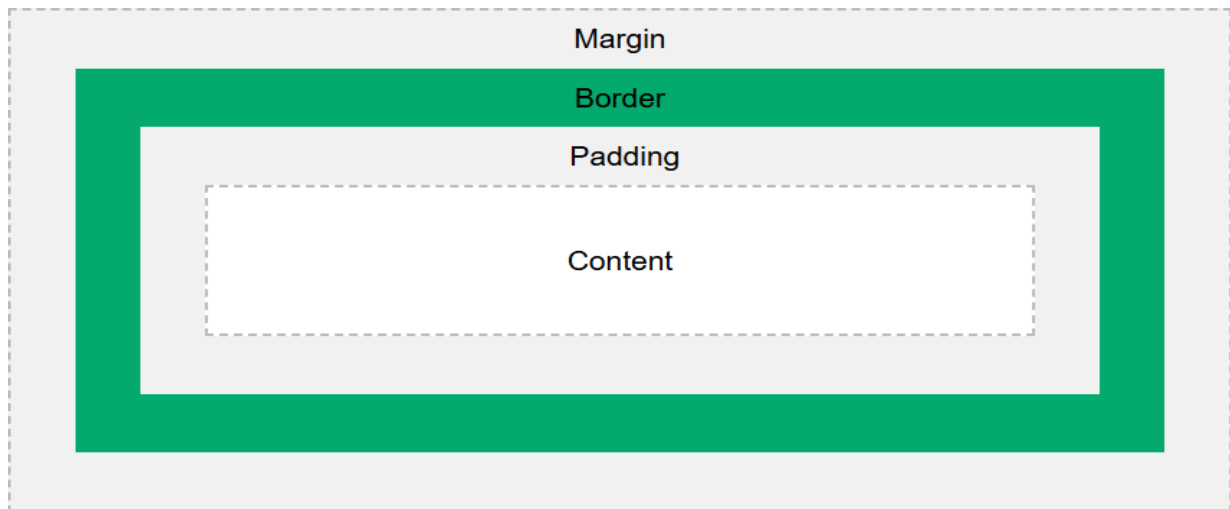
The Text Is Styled Using Font Size, Font Weight, Font Style, Text Decoration, Text Transformation, And Text Alignment.

- d. Write a program, to explain the importance of CSS Box model using**
i. Content ii. Border iii. Margin iv. Padding

CSS Box Model:

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

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



1. **Content** : The content of the box, where text and images appear

Example:

```
div {  
  content: "This is the actual content"; /* This is just an illustration; content is added in HTML. */  
}
```

2. **Padding**: Clears an area around the content. The padding is transparent. Padding can be applied uniformly on all sides or can be customized individually for each side (top, right, bottom, left).

Example1:

```
div {  
  padding: 20px;  
}
```

Example2:

```
div {  
  padding-top: 20px;  
  padding-right: 10px;  
  padding-bottom: 20px;
```

```
padding-left: 10px;
}
```

- 3. Border:** A border that goes around the padding and content. Customize the border with different styles (solid, dashed, dotted), widths, and colors.

Example1:

```
div {
    border: 5px solid black;
}
```

Example2:

```
div {
    border-top: 5px solid red;
    border-right: 3px dotted blue;
}
```

- 4. Margin** - Clears an area outside the border. The margin is transparent.

Example1:

```
div {
    margin: 30px;
}
```

Example2:

```
div {
    margin-top: 10px;
    margin-right: 20px;
    margin-bottom: 30px;
    margin-left: 40px;
}
```

SOURCE CODE: (boxmodels.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>CSS Box Model</title>
<style>
body {
font-family: Arial, sans-serif;
margin: 20px;
background-color: #f0f0f0;
```

```

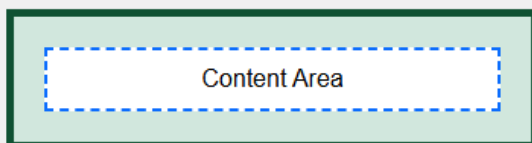
}
.box {
width: 300px;
background-color: #d1e7dd;
border: 5px solid #0f5132;
padding: 20px;
margin: 30px;
}
.content {
background-color: #fff;
text-align: center;
padding: 10px;
border: 2px dashed #0d6efd;
}
</style>
</head>
<body>
<h1>CSS Box Model Example</h1>
<p>The following example shows the different parts of the CSS box model: content, padding,
border, and margin.</p>
<div class="box">
<div class="content">
Content Area
</div>
</div>
</body>
</html>

```

OUTPUT:

CSS Box Model Example

The following example shows the different parts of the CSS box model: content, padding, border, and margin.



EXERCISE-6

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

a. Write a program to embed internal and external JavaScript in a web page.

JavaScript:

- JavaScript (JS) is a **programming language** used for making web pages interactive. It runs in web browsers and allows developers to create dynamic content, control multimedia, animate elements, and handle user interactions.
- It can update and change both HTML and CSS.
- It can calculate, manipulate and validate data.

Internal JavaScript:

Internal JavaScript refers to embedding JavaScript code directly within the HTML file using <script> tag, either inside the <head> or <body>_tag. This method is useful for small scripts specific to a single page.

Syntax:

```
<script>
    // JavaScript code
</script>
```

External JavaScript:

External JavaScript is when the JavaScript code written in another file having an extension .js is linked to the HTML with the src attribute of script tag.

Syntax:

```
<script src="file_name.js"> </script>
```

SOURCE CODE: (external.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Embed JavaScript</title>

    <!-- Internal JavaScript -->

    <script>

        function internalFunction() {
```

```

        alert("Hello from Internal JavaScript!");
    }
</script>
</head>
<body>
    <h1>JavaScript Embedding Example</h1>

    <button onclick="internalFunction()">Run Internal JS</button>
    <button onclick="externalFunction()">Run External JS</button>

    <!-- External JavaScript -->
    <script src="script.js"></script>
</body>
</html>

```

EXTERNAL JS: (external.js)

```

// External JavaScript
function externalAlert() {
    alert("Hello from External JavaScript!");
}

```

OUTPUT:



b. Write a program to explain the different ways for displaying output.

Output Methods in JavaScript:

JavaScript provides multiple ways to display output. Each method serves different use cases, such as debugging, displaying messages to users, or modifying the webpage dynamically.

1. Using console.log():

- Logging Output to Console.
- Best for debugging.
- Not visible to users.

Example:

```
console.log("This is a console log!");
```

2. Using document.write():

- Writing Directly to the Document.
- Quick way to display output.
- Overwrites the entire page if called after loading.

Example:

```
document.write("<h3>Welcome to JavaScript!</h3>");
```

3. Using alert():

- Displaying Popup Alerts.
- Great for small notifications.
- Blocks interaction until dismissed.

Example:

```
alert("This is an alert box!");
```

4. Using innerHTML:

- Modifying HTML Content Dynamically.
- Best for updating content without page reload.
- Can overwrite existing content inside elements.

Example:

```
<p id="message"></p>
<script> document.getElementById("message").innerHTML = "This text is added
dynamically!";
</script>
```

SOURCE CODE: (DifferentWays.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>JavaScript Output Methods</title>

</head>

<body>

  <h2>JavaScript Output Methods</h2>

  <p id="output"></p>

  <script>

    // 1. Using console.log()
```



```

console.log("This is displayed in the console");
// 2. Using document.write()

document.write("<p>This is written using document.write()</p>");

// 3. Using alert()

alert("This is an alert box!");

// 4. Using innerHTML

document.getElementById("output").innerHTML = "This is inserted using innerHTML";

</script>

</body>

</html>

```

OUTPUT:



c. Write a program to explain the different ways for taking input.

Different Ways to Take Input in JavaScript:

JavaScript provides multiple ways to take user input, depending on the use case.

1. Using prompt():

- Simple User Input Dialog
- Quick and easy
- Blocks interaction until user responds

Example :(Taking a Number Input)

```
name = prompt("Enter your name:"); alert("Hello, " + name + "!");
```

2. Using confirm():

- Getting Yes/No Input
- Best for confirmations
- Only returns **true** or **false**

Example:

```

isSure = confirm("Are you sure?");
if (isSure)
{
    console.log("User clicked OK!");
} else {
    console.log("User clicked Cancel!");
}

```

3. Using HTML Form Elements:

- Great for structured input

- Requires additional event handling

Example: (Text Input Field)

```
<input type="text" id="userInput" placeholder="Enter your name">
<button onclick="getInput()">Submit</button>
<p id="output"></p>
<script> function getInput() {
  input = document.getElementById("userInput").value;
  document.getElementById("output").innerText = "You entered: " + input;
}
</script>
```

4. Using addEventListener():

- for Real-time Input
- Captures input dynamically
- More code required than prompt()

Example: (Live Input Display)

```
<input type="text" id="liveInput" placeholder="Type something...">
<p id="liveOutput"></p>
<script>
document.getElementById("liveInput").addEventListener("input", function() {
  document.getElementById("liveOutput").innerText = "You typed: " + this.value; });
</script>
```

5. Using window.event with Key Events:

- Detects key presses
- Not ideal for collecting full input

Example: (Detect Key Presses)

```
<input type="text" id="keyInput" placeholder="Press a key...">
<p id="keyOutput"></p>
<script> document.getElementById("keyInput").addEventListener("keydown",
function(event) { document.getElementById("keyOutput").innerText = "You
pressed: " + event.key; });
</script>
```

6. Using fetch():

- to Get Input from an API
- Great for getting external data
- Requires internet connection

Example: (Fetching JSON Data)

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log("Received data:", data))
  .catch(error => console.error("Error fetching data:", error));
```

SOURCE CODE: (differentinputs.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Taking Input Methods</title>
  <script>
    function takeInput() {
      // Using prompt
      var name = prompt("Enter your name:");

      // Using a form input field
      var age = document.getElementById("ageInput").value;

      alert("Name: " + name + ", Age: " + age);
    }
  </script>
</head>
<body>

  <h1>Taking Input</h1>
  <button onclick="takeInput()">Click to Input Data</button><br><br>
  <!-- Using form input -->
  <input type="text" id="ageInput" placeholder="Enter your age">
</body>
</html>
```

OUTPUT:



d. 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.

SOURCE CODE: (voters.html)

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

```

<title>Voter Information</title>
<script>
    function checkVoterEligibility() {
        var name = prompt("Please enter your name:");
        var age = prompt("Please enter your age:");

        // Convert age to number
        age = parseInt(age);

        // Check voting eligibility
        var eligibility = (age >= 18) ? "Eligible to Vote" : "Not Eligible to Vote";

        // Display in table format
        var table = document.getElementById("voterTable");
        var row = table.insertRow();
        row.insertCell(0).textContent = name;
        row.insertCell(1).textContent = age;
        row.insertCell(2).textContent = eligibility;
    }
</script>
</head>
<body>

<h1>Voter Information</h1>
<button onclick="checkVoterEligibility()">Check Voter Eligibility</button><br><br>

<table border="1" id="voterTable">
    <tr>
        <th>Name</th>
        <th>Age</th>
        <th>Eligibility</th>
    </tr>
</table>

</body>
</html>

```

OUTPUT:

Voter Information

Check Voter Eligibility

Name	Age	Eligibility
Thanu	13	Not Eligible to Vote
Siri	10	Not Eligible to Vote
Harsha	18	Eligible to Vote

EXERCISE-7

7. Java Script Pre-defined and User-defined Objects

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

Document object:

- The document object represents the entire webpage.
- It provides properties and methods to access and modify the content, structure, and style of a webpage dynamically using JavaScript.

Properties:

- **document.title** – Gets or sets the title of the document.
- **document.URL** – Returns the URL of the document.
- **document.body** – Returns the <body> element of the document.
- **document.head** – Returns the <head> element of the document.
- **document.documentElement** – Returns the <html> element of the document.
- **document.doctype** – Returns the Document Type Declaration (DTD) of the document.

Methods:

- **getElementById(id)** – Returns the element with the specified ID.
- **getElementsByClassName(className)** – Returns a collection of elements with the specified class name.
- **getElementsByTagName(tagName)** – Returns a collection of elements with the specified tag name.
- **createElement(tagName)** – Creates a new HTML element with the specified tag name.
- **createTextNode(text)** – Creates a new text node with the specified text.
- **appendChild(node)** – Appends a node as the last child of a node.
- **removeChild(node)** – Removes a child node from the DOM.
- **setAttribute(name, value)** – Sets the value of an attribute on the specified element.
- **getAttribute(name)** – Returns the value of the specified attribute on the element.

SOURCE CODE: (objectmethods.html)

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>

  <meta charset="UTF-8">

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

  <title>Object Document</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      margin: 20px;

    }

    #header {

      padding: 20px;

      text-align: center;

      font-size: 24px;

      background-color: #f4f4f4;

      border: 1px solid #ccc;

    }

    button {

      padding: 10px 20px;

      margin: 5px;

      cursor: pointer;

    }

  </style>

</head>

<body>

  <div id="header">Welcome to the Object Document Example</div>

  <button onclick="changeTitle()">Change Document Title</button>

  <button onclick="changeHeaderColor()">Change Header Color</button>

  <button onclick="showDocumentInfo()">Show Document Info</button>
```

```

<div id="info"></div>

<script>

    // Function to change the document title

    function changeTitle() {

        document.title = "New Document Title-Object document Properties & Methods";

        alert("Document title has been changed to: " + document.title);

    }

    // Function to change the header background color

    function changeHeaderColor() {

        const header = document.getElementById('header');

        header.style.backgroundColor = "lightgreen";

        alert("Header color changed!");

    }

    // Function to display document information

    function showDocumentInfo() {

        const infoDiv = document.getElementById('info');

        infoDiv.innerHTML = `

            <p><strong>Document Title:</strong> ${document.title}</p>

            <p><strong>Document URL:</strong> ${document.URL}</p>

        `;

    }

</script>

</body>

</html>    document.body.appendChild(para);

    // Display the current URL

    console.log("Current URL: " + document.URL);

    // Change the content of an element by its ID

    document.getElementById("demo").textContent = "This is changed using getElementById.";

```

```
</script>
</body>

</html>
```

OUTPUT:



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

Window object:

The **window** object is the global object in JavaScript when running in a web browser, meaning it represents the entire browser window. It provides properties and methods to interact with the browser's environment, including window size, navigation, alerts, and more. It is also the interface for interacting with the document (through the document object), the browser's history, location, and much more.

Properties:

- **window.innerWidth** – Returns the width of the browser window's content area (in pixels).
- **window.innerHeight** – Returns the height of the browser window's content area (in pixels).
- **window.outerWidth** – Returns the total width of the browser window including toolbars, scrollbars, etc.
- **window.outerHeight** – Returns the total height of the browser window including toolbars, scrollbars, etc.
- **window.location** – Provides information about the current URL and allows URL manipulation.
- **window.history** – Gives access to the browser's session history (pages visited in the tab).
- **window.document** – Represents the currently loaded HTML document.
- **window.navigator** – Contains information about the browser (name, version, platform, etc.).

Methods:

- **window.alert()** – Displays an alert dialog with a message.

- **window.confirm()** – Displays a dialog box with "OK" and "Cancel" buttons.
- **window.prompt()** – Displays a dialog box that prompts the user to enter a value.
- **window.open()** – Opens a new browser window or tab.
- **window.close()** – Closes the current browser window (only works on windows opened via window.open()).
- **window.setTimeout()** – Executes a function after a specified delay (in milliseconds).
- **window.clearTimeout()** – Cancels a timeout previously set with setTimeout().

SOURCE CODE: (windowobject.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Window Object Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
      color: deeppink
    }
    button {
      padding: 10px 20px;
      margin: 10px;
      cursor: pointer;
      font-size: 16px;
    }
    button:hover {
      background-color: #45a049; /* Darker background color when the button is hovered */
    }
    #info {
```

```

        margin-top: 20px;
    }
</style>
</head>
<body>
    <h1>Explore the Window Object in JavaScript</h1>
    <button onclick="showWindowSize()">Show Window Size</button>
    <button onclick="showAlert()">Show Alert</button>
    <button onclick="changeLocation()">Change URL</button>
    <button onclick="openNewWindow()">Open New Window</button>
    <button onclick="closeCurrentWindow()">Close Current Window</button>
    <div id="info"></div>
    <script>
        // Show the window size (inner width and height)
        function showWindowSize() {
            const width = window.innerWidth;
            const height = window.innerHeight;
            document.getElementById('info').innerHTML = `
                <p><strong>Window Size:</strong></p>
                <p>Width: ${width}px</p>
                <p>Height: ${height}px</p>
            `;
        }
        // Show an alert box
        function showAlert() {
            window.alert("This is an alert message from the window object!");
        }
        // Change the location (URL) of the browser
        function changeLocation() {
            window.location.href = "https://www.example.com"; // Redirect to a new URL
        }
    </script>
</body>
</html>

```

```
function openNewWindow() {  
    window.open("https://www.example.com", "_blank", "width=600,height=400");  
    document.getElementById('info').innerHTML = "<p>A new window has been opened!</p>";  
}  
  
// Close the current window (this will only work if opened via JavaScript)  
  
function closeCurrentWindow() {  
    if (window.confirm("Are you sure you want to close this window?")) {  
        window.close();  
    }  
}  
  
</script>  
</body>  
</html>
```

OUTPUT:

Explore the Window Object in JavaScript

Show Window Size

Show Alert

Change URL

Open New Window

Close Current Window

c. Write a program using array object properties and methods.

Array object:

An **array object** in JavaScript is a special kind of object used to store an ordered collection of values. Arrays are one of the most commonly used data structures, and they are particularly useful when you need to store and manipulate a list of values that can be of any type, including numbers, strings, objects, or even other arrays.

Properties:

- **length:** Represents the number of elements in the array. It updates automatically when elements are added or removed.

Methods:

- **push():** Adds one or more elements to the end of the array and returns the new length of the

array.

- **pop()**: Removes the last element from the array and returns it.
- **join()**: Joins all elements of the array into a single string, with a specified separator.
- **forEach()**: Executes a provided function once for each element in the array
- **reduce()**: Executes a reducer function on each element in the array (from left to right) to reduce it to a single value.
- **shift()**: Removes the first element from the array and shifts the remaining elements to the left.
- **unshift()**: Adds one or more elements to the beginning of the array and returns the new length.

SOURCE CODE: (arrayobject.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Array Manipulation Example</title>

  <style>

    body {

      font-family: Arial, sans-serif;
      margin: 20px;
    }

    button {

      padding: 10px;
      margin: 5px;
    }

    .output {

      margin-top: 20px;
    }

  </style>

</head>
<body>

  <h1>Array Manipulation Example</h1>
```

```

<h2>Initial Array</h2>

<pre id="initial-array">[10, 20, 30, 40]</pre>

<h2>Manipulate Array</h2>

<button onclick="addElement()">Add Element (50)</button>

<button onclick="removeLastElement()">Remove Last Element</button>

<button onclick="removeFirstElement()">Remove First Element</button>

<button onclick="doubleNumbers()">Double Numbers</button>

<button onclick="filterStrings()">Filter Strings</button>

<div class="output">

  <h3>Output:</h3>

  <pre id="output"></pre>

</div>

<script>

  // Initial array

  let arr = [10, 20, 30, 40];

  // Display the initial array on the page

  document.getElementById("initial-array").textContent = JSON.stringify(arr);

  // Add element 50 to the array

  function addElement() {

    arr.push(50);

    updateOutput();

  }

  // Remove the last element of the array

  function removeLastElement() {

    arr.pop();

    updateOutput();

  }

```

```

// Remove the first element of the array

function removeFirstElement() {

    arr.shift();

    updateOutput();

}

// Double all numbers in the array

function doubleNumbers() {

    arr = arr.map(item => (typeof item === 'number' ? item * 2 : item));

    updateOutput();

}

// Filter out only strings from the array

function filterStrings() {

    let filteredArr = arr.filter(item => typeof item === 'string');

    document.getElementById("output").textContent = "Filtered Strings: " +
JSON.stringify(filteredArr);

}

// Update the output display

function updateOutput() {

    document.getElementById("output").textContent = "Updated Array: " + JSON.stringify(arr);

}

</script>

</body>

</html>

```

OUTPUT:

Array Manipulation Example

Initial Array

[10,20,30,40]

Manipulate Array

Add Element (50)

Remove Last Element

Remove First Element

Double Numbers

Filter Strings

Output:

Updated Array: [10,20,30,40,50]

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

Math Object:

The **Math object** in JavaScript is a built-in object that provides a set of mathematical functions and constants. It's important to note that the Math object itself is not a constructor, meaning you don't create instances of Math; instead, you directly use its properties and methods.

Properties:

- **Math.PI:** The constant π (Pi), approximately 3.14159. It's used for calculations involving circles.
- **Math.E:** The constant e, Euler's number, approximately 2.71828. It's the base of natural logarithms.
- **Math.LN10:** The natural logarithm of 10, approximately 2.30258.
- **Math.LN2:** The natural logarithm of 2, approximately 0.69314.
- **Math.LOG10E:** The base-10 logarithm of e, approximately 0.43429.
- **Math.SQRT2:** The square root of 2, approximately 1.41421.
- **Math.MAX_VALUE:** The largest positive number that can be represented in JavaScript.
- **Math.MIN_VALUE:** The smallest positive number that can be represented in JavaScript.

Methods:

- **Math.abs(x):** Returns the absolute value of a number, i.e., removes the sign.
- **Math.round(x):** Rounds a number to the nearest integer. If the fractional part is 0.5 or greater, it rounds up.
- **Math.floor(x):** Rounds a number **down** to the nearest integer.
- **Math.ceil(x):** Rounds a number **up** to the nearest integer.
- **Math.max(x, y, ...):** Returns the largest of the given numbers.
- **Math.min(x, y, ...):** Returns the smallest of the given numbers.
- **Math.random():** Returns a pseudo-random floating point number between 0 (inclusive) and 1 (exclusive).
- **Math.pow(x, y):** Returns the value of x raised to the power of y.
- **Math.sqrt(x):** Returns the square root of a number x.

SOURCE CODE: (mathobject.html)

```
<!DOCTYPE html>
```

```

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Math Operations with JavaScript</title>

  <style>

    body {

      font-family: Arial, sans-serif;
      margin: 20px;
    }

    label, input, button {
      margin: 5px;
      padding: 10px;
    }

    .output {

      margin-top: 20px;
      font-size: 18px;
      font-weight: bold;
    }

    select {

      padding: 10px;
      font-size: 16px;
    }

  </style>

</head>

<body>

  <h1>Math Operations with JavaScript</h1>

  <label for="operation">Choose a math operation:</label>

  <select id="operation">
    <option value="sqrt">Square Root</option>

    <option value="pow">Power</option>

    <option value="log">Natural Logarithm</option>
  </select>

```



```

    <option value="log10">Logarithm Base 10</option>

    <option value="random">Random Number</option>
</select>

<div id="inputs">

    <label for="number1">Enter a number:</label>

    <input type="number" id="number1" placeholder="Number" />

    <label for="number2" id="label-number2" style="display: none;">Enter second number (for
power operation):</label>

    <input type="number" id="number2" placeholder="Second number" style="display: none;" />
</div>

<button onclick="performOperation()">Calculate</button>

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

<script>

    // Function to perform the selected math operation

    function performOperation() {

        const operation = document.getElementById("operation").value;

        const num1 = parseFloat(document.getElementById("number1").value);

        const num2 = parseFloat(document.getElementById("number2").value);

        let result;

        // Validate the input

        if (isNaN(num1) || (operation === "pow" && isNaN(num2))) {

            document.getElementById("result").textContent = "Please enter valid numbers.";

            return;

        }

        switch (operation) {

            case "sqrt":

                result = Math.sqrt(num1);

                break;

```

```

case "pow":
    result = Math.pow(num1, num2);
    break;
case "log":
    if (num1 <= 0) {
        result = "Logarithm is undefined for non-positive numbers.";
    } else {
        result = Math.log(num1);
    }
    break;
case "log10":
    if (num1 <= 0) {
        result = "Logarithm (base 10) is undefined for non-positive numbers.";
    } else {
        result = Math.log10(num1);
    }
    break;
case "random":
    result = Math.random();
    break;
default:
    result = "Invalid operation.";
    break;
}

// Display the result
document.getElementById("result").textContent = `Result: ${result}`;
}

```

```
// Update the input fields based on selected operation

document.getElementById("operation").addEventListener("change", function() {

    const operation = this.value;

    const number2Input = document.getElementById("number2");

    const labelNumber2 = document.getElementById("label-number2");

    if (operation === "pow") {

        number2Input.style.display = "inline-block";

        labelNumber2.style.display = "inline-block";

    } else {

        number2Input.style.display = "none";

        labelNumber2.style.display = "none";

    }

});

</script>

</body>

</html>
```

OUTPUT:

Math Operations with JavaScript

Choose a math operation:

Square Root



Enter a number:

225

Calculate

Result: 15

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

Regex Object:

In JavaScript, regular expressions (regex) are represented by the RegExp object. This object provides several properties and methods that allow you to work with patterns for string matching and manipulation.

Properties:

- **lastIndex:** This property is used to specify the index at which to start the next match. When a regular expression is used with methods like `exec()` or `test()`, this property will automatically update after each match.
- **source:** A string representation of the pattern of the regular expression. This is the text of the regex without any flags (e.g., `/a/g` becomes `"a"`).
- **flags:** This property returns the flags used in the regular expression (such as `"g"`, `"i"`, `"m"`).
- **global** (only available as a boolean in newer engines): Returns true if the regular expression has the `g` flag, indicating that it performs global searches.
- **ignoreCase** (only available as a boolean in newer engines): Returns true if the regular expression has the `i` flag, meaning the search is case-insensitive.
- **multiline** (only available as a boolean in newer engines): Returns true if the regular expression has the `m` flag, indicating it allows the `^` and `$` anchors to match at the start or end of each line, rather than just at the start and end of the entire string.

Methods:

- **exec():** Executes a search for a match in a string and returns an array of matches (or null if no match is found). It also updates the `lastIndex` property.
- **test():** Tests whether a string matches the pattern of the regular expression. Returns true if there's a match, otherwise false.
- **toString():** Returns a string representing the regular expression, including the delimiters and any flags. Essentially, it outputs the regular expression in the form `/pattern/flags`.
- **compile():** Re-compiles the regular expression with a new pattern and optional flags. This allows you to modify an existing regex object.
- **dotAll** (added in ECMAScript 2018): If the `s` flag is set, it allows the dot (`.`) to match newline characters as well.

SOURCE CODE: (regexobject.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Regex Validation Example</title>

  <style>

    body {

      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;

      background-color: #f4f4f9;

    }

    .container {

      max-width: 500px;

      margin: 50px auto;

      padding: 20px;

      background-color: #fff;

      border-radius: 8px;

      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

    }

    h1 {

      text-align: center;

      color: #333;

    }

    input[type="email"] {

      width: 100%;
```

```
padding: 10px;

margin: 10px 0;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

}

button {

width: 100%;

padding: 10px;

background-color: #4CAF50;

color: white;

border: none;

border-radius: 4px;

font-size: 16px;

}

button:hover {

background-color: #45a049;

}

.message {

text-align: center;

margin-top: 20px;

}

.valid {

color: green;

}

.invalid {

color: red;

}
```

```

</style>

</head>

<body>

<div class="container">

  <h1>Email Validation</h1>

  <form id="emailForm">

    <input type="email" id="email" placeholder="Enter your email" required>

    <button type="submit">Submit</button>

  </form>

  <div class="message" id="message"></div>

</div>

<script>

  // Regular expression for email validation

  const emailRegex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;

  // Getting form and message elements

  const form = document.getElementById('emailForm');

  const emailInput = document.getElementById('email');

  const message = document.getElementById('message');

  // Event listener for form submission

  form.addEventListener('submit', function(event) {

    event.preventDefault(); // Prevent the form from refreshing the page

    // Get the email value

    const email = emailInput.value;

    // Check if the email matches the regex

    if (emailRegex.test(email)) {

      message.textContent = 'Email is valid!';

```

```

        message.classList.remove('invalid');

        message.classList.add('valid');

    } else {

        message.textContent = 'Invalid email address. Please try again.';

        message.classList.remove('valid');

        message.classList.add('invalid');

    }

});

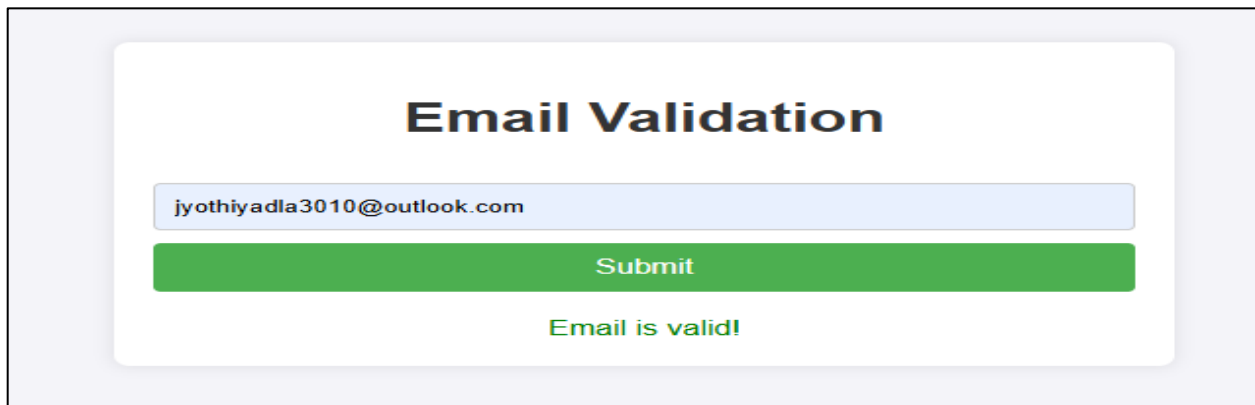
</script>

</body>

</html>

```

OUTPUT:



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

String Object:

In JavaScript, strings are primitive data types, but they also have many built-in properties and methods provided by the String object. These properties and methods allow you to manipulate and work with text data efficiently.

Properties:

- **length:** This property returns the number of characters in the string.
- **constructor:** This property returns a reference to the String constructor function that created the string.

- **prototype**: This property allows you to extend the functionality of strings by adding custom methods or properties. It is usually used when creating custom string-like objects.

Methods:

- **charAt(index)**: Returns the character at the specified index.
- **charCodeAt(index)**: Returns the Unicode (UTF-16) code of the character at the specified index.
- **concat(str1, str2, ...)**: Joins two or more strings together.
- **includes(substring)**: Checks if the string contains the specified substring. Returns true or false.
- **indexOf(substring)**: Returns the index of the first occurrence of the specified substring. If not found, returns -1.
- **lastIndexOf(substring)**: Returns the index of the last occurrence of the specified substring.
- **match(regex)**: Searches the string for a match against a regular expression and returns an array of matches (or null if no match is found).
- **replace(regex|substr, newSubstr)**: Replaces occurrences of a specified substring or regular expression with a new substring.
- **search(regex)**: Searches for a match against a regular expression. Returns the index of the first match or -1 if no match is found.
- **slice(start, end)**: Extracts a part of a string and returns the extracted part.
- **toLowerCase()**: Converts the string to lowercase.
- **toUpperCase()**: Converts the string to uppercase.

SOURCE CODE: (stringobject.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

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

  <title>Interactive String Methods Explorer</title>

  <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
rel="stylesheet">

</head>

<body class="bg-gradient-to-r from-purple-400 via-pink-500 to-red-500 flex items-center justify-
center min-h-screen">
```

```

<div class="bg-white p-8 rounded-xl shadow-xl w-96 text-center">

  <h2 class="text-3xl font-extrabold mb-6 text-gray-800">String Methods Explorer</h2>

  <input id="inputString" type="text" placeholder="Enter a string here" class="w-full p-3 border-2
border-gray-300 rounded-md mb-4 focus:outline-none focus:ring-2 focus:ring-purple-500" />

  <button onclick="exploreMethods()" class="w-full bg-purple-600 text-white p-3 rounded-md
hover:bg-purple-700 transition duration-300">Explore String Methods</button>

  <div id="output" class="mt-6 text-left text-gray-700 space-y-2"></div>

</div>

<script>

function exploreMethods() {

  const str = document.getElementById('inputString').value;

  const output = `

    <p><strong>Original String:</strong> ${str}</p>

    <p><strong>Length:</strong> ${str.length}</p>

    <p><strong>Uppercase:</strong> ${str.toUpperCase()}</p>

    <p><strong>Lowercase:</strong> ${str.toLowerCase()}</p>

    <p><strong>Trimmed:</strong> ${str.trim()}</p>

    <p><strong>Reversed:</strong> ${str.split('').reverse().join('')}</p>

    <p><strong>First Char:</strong> ${str.charAt(0)}</p>

    <p><strong>Includes 'a'?</strong> ${str.includes('a')}</p>

    <p><strong>Slice (0,3):</strong> ${str.slice(0,3)}</p>

  `;

  document.getElementById('output').innerHTML = output;

}

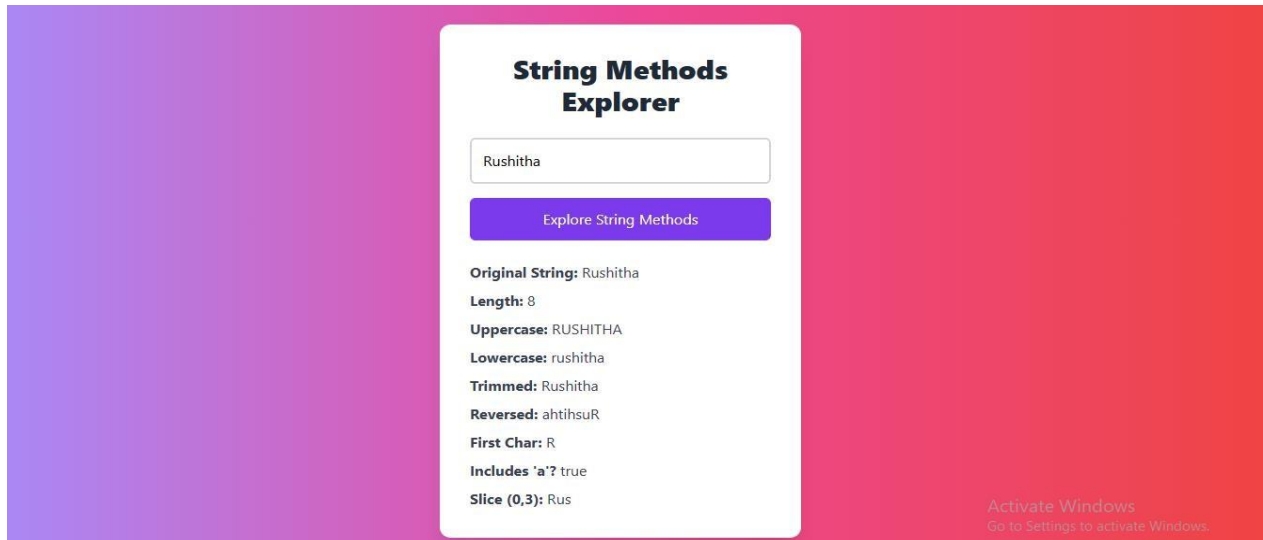
</script>

</body>

</html>

```

OUTPUT:



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

Date object:

The Date object in JavaScript provides methods to work with dates and times. It allows you to create, manipulate, and format dates easily.

Properties:

- **Date.length** – Always returns 7 as the number of arguments for the constructor.

Methods:

- **getMinutes():** Returns the minutes (0-59)
- **getSeconds():** Returns the seconds (0-59).
- **getTime():** Returns the number of milliseconds since Jan 1, 1970 (epoch time).
- **toDateSting():** Returns the date portion as a readable string.
- **setDate(day):** Sets the day of the month for a date object.
- **setFullYear(year, month, day):** Sets the year, month, and optionally the day.
- **setMonth(month):** Sets the month (0-11).
- **setHours(hours, minutes, seconds, ms):** Sets the hour, and optionally minutes, seconds, and milliseconds.

-

SOURCE CODE: (dateobject.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

<meta charset="UTF-8" />

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

<title>Date Object Explorer</title>

<link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
rel="stylesheet">

</head>

<body class="bg-gradient-to-br from-blue-500 to-green-400 min-h-screen flex items-center justify-
center">

  <div class="bg-white p-8 rounded-xl shadow-xl w-full max-w-lg text-center">

    <h2 class="text-3xl font-extrabold mb-6 text-gray-800">JavaScript Date Object Explorer</h2>

    <button onclick="showDateInfo()" class="mb-4 px-5 py-3 bg-blue-600 text-white rounded-lg
hover:bg-blue-700 transition">Show Current Date Info</button>

    <div id="dateOutput" class="text-left text-gray-700 space-y-2"></div>

  </div>

  <script>

function showDateInfo() {

  const now = new Date();

  const output = `

    <p><strong>Current Date and Time:</strong> ${now.toString()}</p>

    <p><strong>ISO Format:</strong> ${now.toISOString()}</p>

    <p><strong>UTC String:</strong> ${now.toUTCString()}</p>

    <p><strong>Locale Date:</strong> ${now.toLocaleDateString()}</p>

    <p><strong>Locale Time:</strong> ${now.toLocaleTimeString()}</p>

    <p><strong>Year:</strong> ${now.getFullYear()}</p>

    <p><strong>Month:</strong> ${now.getMonth() + 1}</p>

    <p><strong>Day:</strong> ${now.getDate()}</p>

    <p><strong>Day of Week:</strong> ${now.getDay()}</p>

    <p><strong>Hours:</strong> ${now.getHours()}</p>

    <p><strong>Minutes:</strong> ${now.getMinutes()}</p>

```

```

<p><strong>Seconds:</strong> ${now.getSeconds()}</p>

<p><strong>Milliseconds:</strong> ${now.getMilliseconds()}</p>

`;

document.getElementById('dateOutput').innerHTML = output;

}

</script>

</body>

</html>

```

OUTPUT:



- h. Write a program to explain user-defined object by using properties, methods, accessors, constructors and display.

User-Defined Object:

A **User-Defined Object** in JavaScript is an object created by the programmer to model real-world entities, store data, and define behaviors through properties and methods.

- To group related data and functions together.
- To create multiple instances of similar objects with ease.
- To achieve better organization, code reuse, and readability.

Properties: Properties are values associated with an object.

Methods: Methods are functions stored as object properties.

Accessors (getters and setters):

- **Getters** retrieve property values.
- **Setters** modify property values

Constructors: A constructor is a function used to create multiple objects with the same structure.

Display: Display objects in various ways: console, DOM, JSON, etc.

SOURCE CODE: (userdefinedobject.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

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

  <title>User-Defined Objects Demo</title>

  <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
rel="stylesheet">

</head>

<body class="bg-gradient-to-r from-indigo-400 to-pink-500 min-h-screen flex items-center justify-
center">

  <div class="bg-white p-8 rounded-xl shadow-2xl w-full max-w-lg text-center">

    <h2 class="text-4xl font-extrabold mb-6 text-gray-800">Object-Oriented JS Demo</h2>

    <button onclick="showPeople()" class="mb-4 px-6 py-3 bg-indigo-600 text-white rounded-lg
hover:bg-indigo-700 transition">Create and Display People</button>

    <div id="output" class="text-left text-gray-700 space-y-3"></div>

  </div>

  <script>

    class Person {

      constructor(firstName, lastName, age) {

        this.firstName = firstName;
```

```

    this.lastName = lastName;

    this.age = age;
  }

  get fullName() {

    return `${this.firstName} ${this.lastName}`;

  }

  set updateAge(newAge) {

    this.age = newAge;

  }

  displayInfo() {

    return `${this.fullName}, Age: ${this.age}`;

  }

}

function showPeople() {

  const people = [

    new Person('Alice', 'Johnson', 28),

    new Person('Bob', 'Smith', 32),

    new Person('Charlie', 'Brown', 25)

  ];

  people[0].updateAge = 29;

  let outputHTML = people.map(person => `

    <div class="p-4 bg-gray-100 rounded-lg shadow-sm">

      <p><strong>Name:</strong> ${person.fullName}</p>

      <p><strong>Age:</strong> ${person.age}</p>

      <p><strong>Info:</strong> ${person.displayInfo()}</p>

    </div>

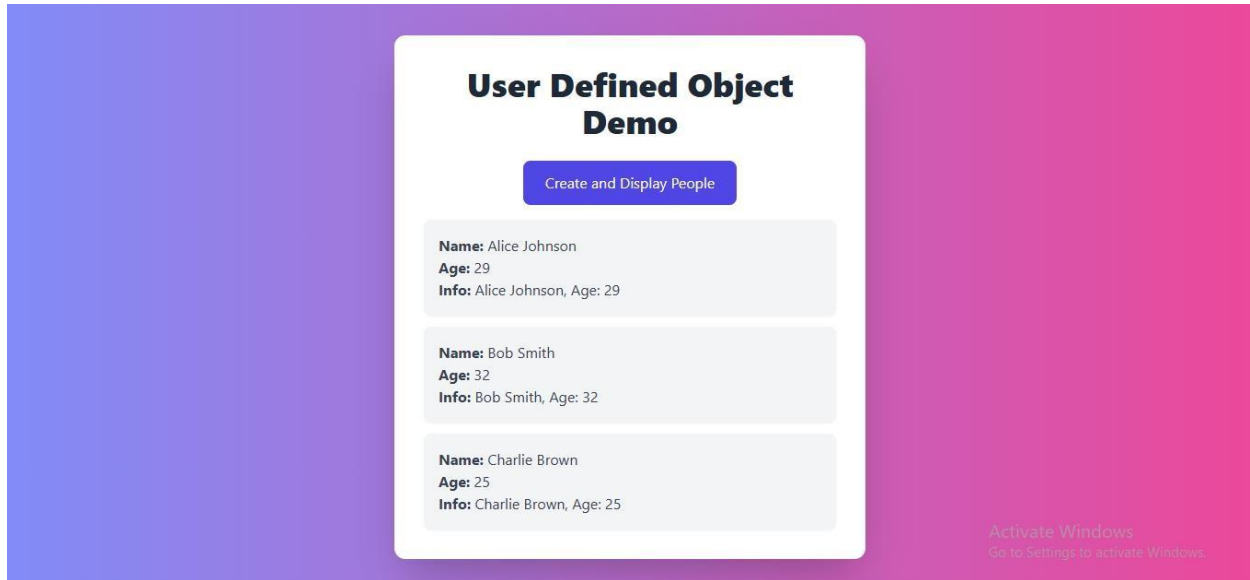
  `).join("");

  document.getElementById('output').innerHTML = outputHTML;

```

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

OUTPUT:



EXERCISE-8

8. Java Script 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”.

SOURCE CODE: (conditionalStatements.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Conditional statements</title>
</head>
<body>
  <h1>Js conditional statements</h1>
  <label>First number:</label>
  <input type="text" id="one"><br><br>
  <label>Second number:</label>
  <input type="text" id="two"><br><br>
  <label>Third number:</label>
  <input type="text" id="three"><br><br>
  <button onclick="comparison()">submit</button>
  <p id="result"></p>

  <!-- java script code -->
  <script>
    function comparison(){
      let firstNmb = parseInt(document.getElementById('one').value);
      let secondNmb = parseInt(document.getElementById('two').value);
      let thirdNmb = parseInt(document.getElementById('three').value);
      if(firstNmb === secondNmb && secondNmb===thirdNmb){
        document.getElementById('result').textContent="Equal  Numbers";
      }
      else if (firstNmb>secondNmb && firstNmb>thirdNmb){
        document.getElementById('result').textContent="Large Number "+firstNmb;
      }
      else if(secondNmb>thirdNmb){
        document.getElementById('result').textContent="Large Number "+secondNmb;
      }
      else{
        document.getElementById('result').textContent="Large Number "+thirdNmb;
      }
    }
  </script>
</body>
</html>
```

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

OUTPUT:

Js conditional statements

First number:

Second number:

Third number:

Large Number 8

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

SOURCE CODE: (week days.html)

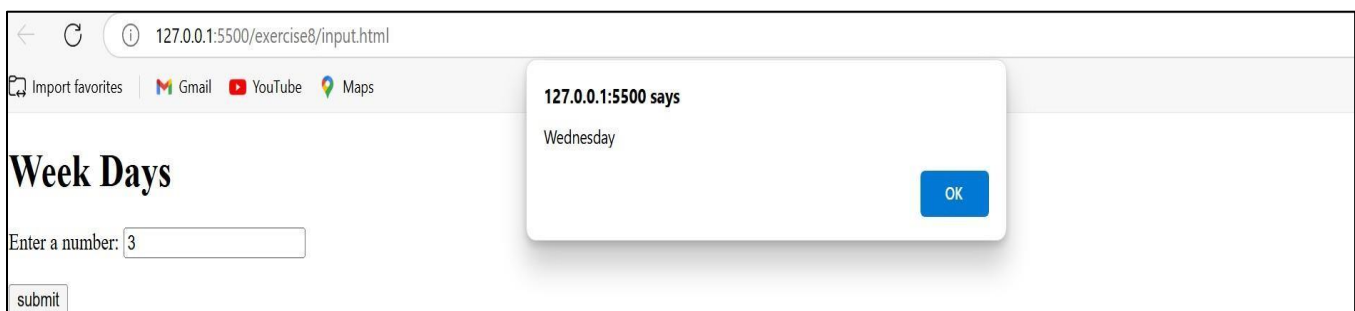
```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Week days</title>  
</head>  
<body>  
  <h1>Week Days</h1>  
  <label>Enter a number:</label>  
  <input type="text" id="week"><br><br>  
  
  <button onclick="week()">submit</button>  
  
  <!-- java script code -->  
  <script>  
    function week(){  
      let weekNmber = parseInt(document.getElementById('week').value);  
  
      switch (weekNmber) {  
        case 1:  
          alert("Monday");  
          break;  
        case 2:  
          alert("Tuesday");  
          break;  
        case 3:  
          alert("Wednesday");
```

```

        break;
    case 4:
        alert("Thursday");
        break;
    case 5:
        alert("Friday");
        break;
    case 6:
        alert("Saturday");
        break;
    case 7:
        alert("Sunday");
        break;
    default:
        alert("Invalid day number. Please enter a number between 1 and 7.");
    }
}
</script>
</body>
</html>

```

OUTPUT:



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

SOURCE CODE: (loops.js)

```

console.log("Using for loop:");
for (let i = 1; i <= 10; i++) {
    console.log(i);
}

console.log("Using while loop:");
let i = 1;
while (i <= 10) {
    console.log(i);
    i++;
}

```

```
console.log("Using do-while loop:");  
let j = 1;  
do {  
  console.log(j);  
  j++;  
} while (j <= 10);
```

OUTPUT:

```
Using for loop:  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
Using while loop:  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
Using do-while loop:  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

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

SOURCE CODE: (AdvLoops.js)

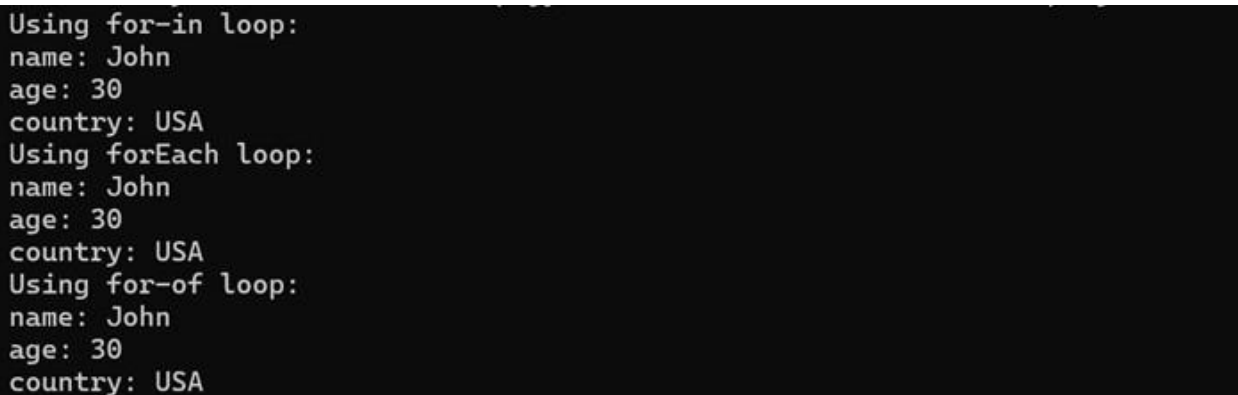
```
let person = {
  name: "John",
  age: 30,
  country: "USA"
};

// Using for-in loop
console.log("Using for-in loop:");
for (let key in person) {
  console.log(key + ": " + person[key]);
}

// Using forEach method (Array format)
console.log("Using forEach loop:");
Object.entries(person).forEach(([key, value]) => {
  console.log(key + ": " + value);
});

// Using for-of loop
console.log("Using for-of loop:");
for (let [key, value] of Object.entries(person)) {
  console.log(key + ": " + value);
}
```

OUTPUT:



```
Using for-in loop:
name: John
age: 30
country: USA
Using forEach loop:
name: John
age: 30
country: USA
Using for-of loop:
name: John
age: 30
country: USA
```

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., $1^3 + 5^3 + 3^3 = 153$]

SOURCE CODE: (Armstrong.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Armstrong</title>
</head>
<body>
  <h1>Armstrong or Not</h1>
  <label>Enter a number:</label>
  <input type="text" id="number"><br><br>

  <button onclick="isArmstrong()">submit</button>

  <!-- java script code -->
  <script>
    function isArmstrong() {
      // Ask the user to input a number
      // Ask user for input
      let num = parseInt(document.getElementById('number').value);

      // Convert the number to a string to get the number of digits
      let numStr = num.toString();
      let length = numStr.length;

      // Initialize a variable to hold the sum of the cubes of the digits
      let sum = 0;

      // Loop through each digit of the number
      for (let i = 0; i < length; i++) {
        let digit = parseInt(numStr[i]); // Get the digit at position i
        sum += Math.pow(digit, length); // Add the power of the digit to the sum
      }

      // Check if the sum equals the original number
      if (sum === num) {
        alert(num + " is an Armstrong number.");
      } else {
        alert(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)

SOURCE CODE: (AmountDenominations.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title> Amount Denominations </title>
</head>
<body>s
  <h1>Amount Denominations</h1>
  <label>Enter a amount:</label>
  <input type="text" id="amount"><br><br>

  <button onclick="calculateDenominations()">submit</button>

  <!-- java script code -->
  <script>
    // Function to calculate denominations
function calculateDenominations() {
  // Ask the user to enter the deposited amount
let amount = parseInt(document.getElementById('amount').value);
  // Ask the user to enter the deposited amount

// Denominations array (100's, 50's, 20's, 10's, 5's, 2's, 1's)
let denominations = [100, 50, 20, 10, 5, 2, 1];
let result = "";

// Loop through each denomination
for (let i = 0; i < denominations.length; i++) {
  let denom = denominations[i];
  let count = Math.floor(amount / denom); // Calculate how many notes/coins of this denomination
  if (count > 0) {
    result += `${count}-${denom}'s, `; // Add to the result string
    amount -= count * denom; // Reduce the amount by the calculated denomination
  }
}

// Output the result (remove the last comma and space)
alert("Denominations: " + result.slice(0, -2));
}
  </script>
</body>
</html>
```

OUTPUT:

Import favorites Gmail YouTube Maps

Amount Denominations

Enter a amount:

127.0.0.1:5500 says

Denominations: 1-100's, 1-50's, 1-10's

EXERCISE-9

9. Java Script Functions and Events

a. Design a 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

SOURCE CODE: (functions.js)

```
// Function to calculate Factorial
function factorial() {
    let num=11
    let result = 1;
    for (let i = 1; i <= num; i++) {
        result *= i;
    }
    return result;
}

// Function to generate Fibonacci series up to n
function fibonacci() {
    let num=11
    let fibSeries = [];
    let a = 0, b = 1;
    while (a <= num) {
        fibSeries.push(a);
        let next = a + b;
        a = b;
        b = next;
    }
    return fibSeries;
}

// Function to check if a number is prime
function primeNumbers() {
    let number=11
    let primes = [];
    for (let num = 2; num <= number; num++) {
        let isPrime = true;
        for (let i = 2; i <= Math.sqrt(num); i++) {
            if (num % i === 0) {
                isPrime = false;
                break;
            }
        }
        if (isPrime) primes.push(num);
    }
}
```

```

    }
    return primes;
}

// Function to check if a number is a palindrome
function isPalindrome() {
    let num=11
    let str = num.toString();
    let reversed = str.split('').reverse().join('');
    return str === reversed;
}

console.log("factorial")
console.log(factorial())
console.log("fibonacci")
console.log(fibonacci())
console.log("prime numbers")
console.log(primeNumbers())
console.log("isPalindrome")
console.log(isPalindrome())

```

OUTPUT:

```

factorial
39916800
fibonacci
[
  0, 1, 1, 2,
  3, 5, 8
]
prime numbers
[ 2, 3, 5, 7, 11 ]
isPalindrome
true

```

- b. 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**

SOURCE CODE: (buttons.html)

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Number Operations</title>
  <script>
    // The JavaScript functions go here (factorial, fibonacci, primeNumbers, isPalindrome)
    // Refer to the previous JavaScript code above.

```

```

function calculateFactorial() {
  let num = parseInt(document.getElementById("num").value); // Get the number from input
  let result = 1;

  // Loop to calculate factorial
  for (let i = 1; i <= num; i++) {
    result *= i;
  }
  alert("Factorial of " + num + " is " + result);
}

function calculateFibonacci() {
  let num = parseInt(document.getElementById("num").value); // Get the number from input
  let a = 0, b = 1;
  let result = "";

  // Fibonacci sequence generation up to the number
  while (a <= num) {
    result += a + ", "; // Append the number to result string
    let next = a + b;
    a = b;
    b = next;
  }
  // Display the Fibonacci sequence
  alert("Fibonacci series up to " + num + ": " + result);
}

function displayPrimeNumbers() {
  let num = parseInt(document.getElementById("num").value); // Get the number from input
  let result = "";
  // Loop to check for prime numbers up to the entered number
  for (let i = 2; i <= num; i++) {
    let isPrime = true;

    // Check if the number is divisible by any number from 2 to sqrt(i)
    for (let j = 2; j <= Math.sqrt(i); j++) {
      if (i % j === 0) {
        isPrime = false;
        break;
      }
    }
    // If it's prime, append it to the result string
    if (isPrime) {
      result += i + ", ";
    }
  }
  // Display the prime numbers
  alert("Prime numbers up to " + num + ": " + result);
}

```

```

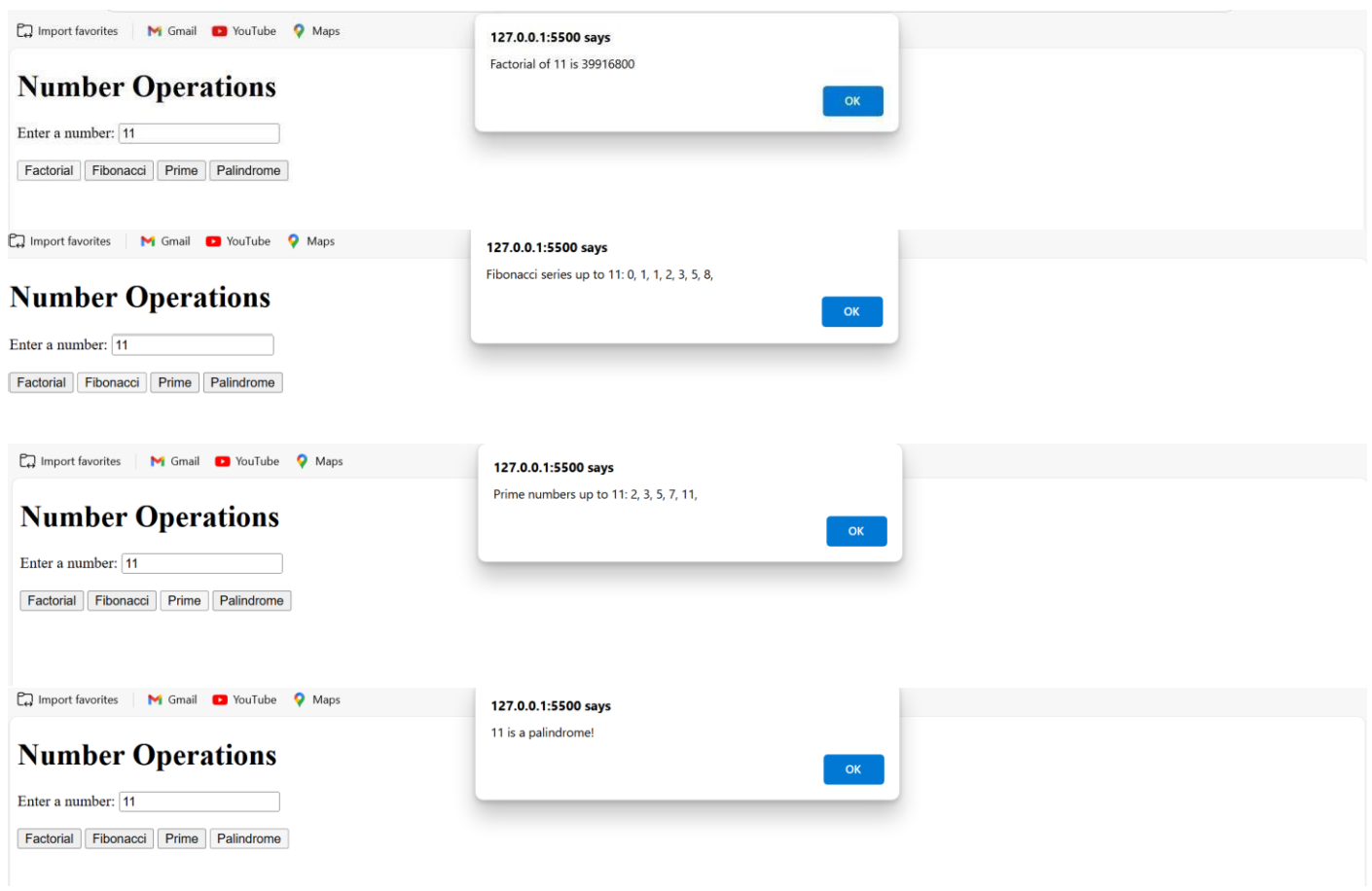
function checkPalindrome() {
    let num = document.getElementById("num").value; // Get the number from input
    let str = num.toString(); // Convert the number to a string
    let reversed = str.split("").reverse().join(""); // Reverse the string

    // Check if the original number matches the reversed string
    if (str === reversed) {
        alert(num + " is a palindrome!");
    } else {
        alert(num + " is not a palindrome.");
    }
}
</script>
</head>
<body>
<h1>Number Operations</h1>
<label for="num">Enter a number:</label>
<input type="text" id="num" />

<br><br>
<button onclick="calculateFactorial()">Factorial</button>
<button onclick="calculateFibonacci()">Fibonacci</button>
<button onclick="displayPrimeNumbers()">Prime</button>
<button onclick="checkPalindrome()">Palindrome</button>
</body>
</html>

```

OUTPUT:



c. 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 (only numbers and length 10 digits) iii. E-mail (should contain format like [xxxxxxx@xxxxxx.xxx](#))

SOURCE CODE: (login.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Registration Page</title>
  <script>
    function validateForm() {
      // Validate Name: Should start with a letter, alphanumeric, and length >= 6
      let name = document.getElementById("name").value;
      let nameRegex = /^[A-Za-z][A-Za-z0-9]{5,}$/;
      if (!nameRegex.test(name)) {
        alert("Invalid name. It must start with a letter, be alphanumeric, and at least 6 characters long.");
        return false;
      }

      // Validate Mobile: Only numbers and length should be 10 digits
      let mobile = document.getElementById("mobile").value;
      let mobileRegex = /^\d{10}$/;
      if (!mobileRegex.test(mobile)) {
        alert("Invalid mobile number. It must be 10 digits.");
        return false;
      }

      // Validate Email: Format like xxxxxxxx@xxxxxx.xxx
      let email = document.getElementById("email").value;

      let emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
      if (!emailRegex.test(email)) {
        alert("Invalid email address. Please follow the format xxxxxxxx@xxxxxx.xxx.");
        return false;
      }

      alert("Registration Successful!");
      return true;
    }
  </script>
</head>
<body>
```

```
<h1>Registration Form</h1>
<form onsubmit="return validateForm()">
  <label for="name">Name:</label>
  <input type="text" id="name" required /><br><br>

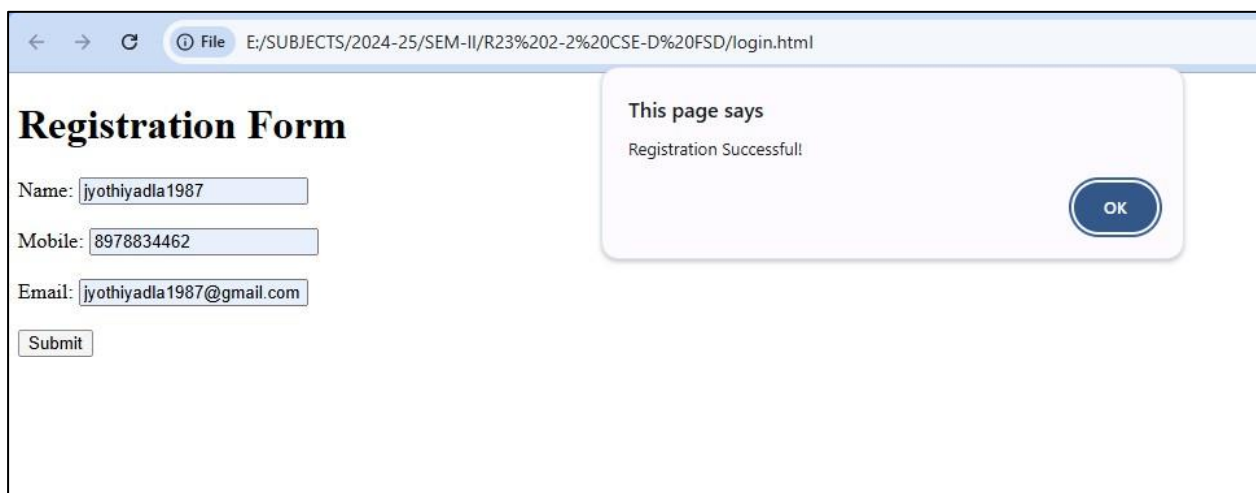
  <label for="mobile">Mobile:</label>
  <input type="text" id="mobile" required /><br><br>

  <label for="email">Email:</label>
  <input type="text" id="email" required /><br><br>

  <button type="submit">Submit</button>
</form>

</body>
</html>
```

OUTPUT:



The screenshot shows a web browser window with the address bar displaying "E:/SUBJECTS/2024-25/SEM-II/R23%202-2%20CSE-D%20FSD/login.html". The page title is "Registration Form". The form contains three input fields: "Name" with the value "jyothiyadla1987", "Mobile" with the value "8978834462", and "Email" with the value "jyothiyadla1987@gmail.com". A "Submit" button is located below the email field. On the right side of the page, there is a light purple box with the text "This page says" and "Registration Successful!". An "OK" button is located at the bottom right of this box.