Expno:4

Aim: Write a program to apply different types of selector froms

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

Description:

A Simple Selector in CSS is a way to target specific HTML elements to apply styles. They form the basic building blocks of any CSS stylesheet.

Here are the five main types of simple selectors:

1. Element Selector (Type Selector)

- Selects all instances of a particular HTML element type.
- Syntax: Just the name of the HTML tag.
- Example:
    - p selects all paragraph elements.
    - h1 selects all level-one heading elements.
    - div selects all div container elements.

2. ID Selector

- Selects a single, unique HTML element based on its id attribute. IDs must be unique within an HTML document.
- Syntax: A hash sign (#) followed immediately by the ID value.
- Example:
    - HTML: ...

3. Class Selector

- Selects all HTML elements that have a specific value in their class attribute. An element can have multiple classes, and multiple elements can share the same class.
- Syntax: A period or dot (.) followed immediately by the class name.
- Example: HTML: ... and ...

4. Group Selector

- Selects multiple distinct selectors (elements, IDs, or classes) and applies the same set of styles to all of them. This is used for efficiency to avoid repeating the same styles for different selectors.
- Syntax: The selectors are separated by a comma (,).
- Example: h1, h2, .title, #logo selects all h1 elements, all h2 elements, all elements with the class title, and the single element with the ID logo, and applies the styles to all four groups.

5. Universal Selector

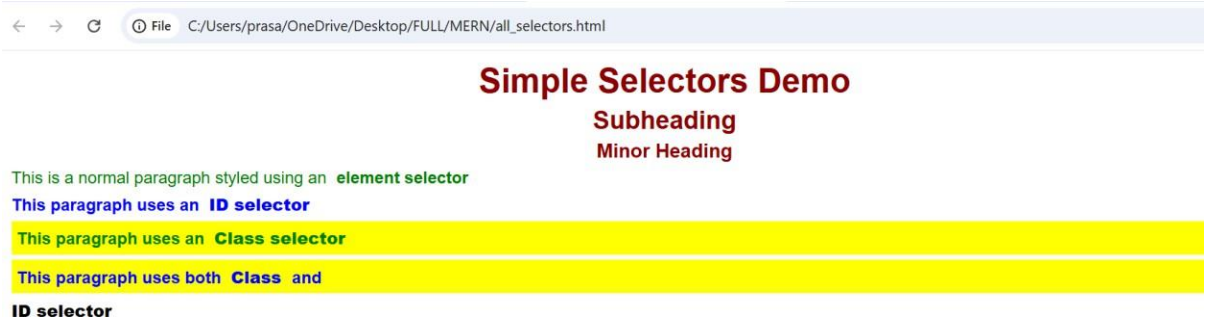
- Selects all HTML elements on the page. It's often used to apply a global style, like resetting margins or borders.
- Syntax: An asterisk (*).
- Example:
    - * selects every single element in the document.
    - div * selects every element that is a descendant of a div.

Program:

```
<html lang="en">
<head>
<title> Simple Selectors example</title>
<style>
/*element selector*/
p{
color:green;
font-size:16px;
}
/* ID Selector*/
#unique{
color:blue;
font-weight:bold;
}
/* Class Selector*/
.highlight{
background-color:yellow;
padding:5px;
}
/* Group Selector*/
h1,h2,h3{
color:darkred;
text-align:center;
```

```
}
/* Universal Selector*/
*{
font-family:Arial,sans-serif;
margin:5px;
}
</style>
</head>
<body>
 <h1> Simple Selectors Demo</h1>
<h2>Subheading</h2>
<h3>Minor Heading</h3>
<p> This is a normal paragraph styled using an<strong> element selector</selector></p>
<p id="unique"> This paragraph uses an <strong> ID selector</strong></p>
<p class="highlight"> This paragraph uses an <strong> Class selector</strong></p>
<p class="highlight" id="unique"> This paragraph uses both <strong> Class </strong> and </p><strong> ID selector </strong></p>
</body>
</html>
```

Output:



Result:

ii) Combinator Selector (descendent , child, adjacent sibiling, genrral sibling)

Description:

A **Combinator Selector** in CSS is used to combine two or more simple selectors in a meaningful way to target elements based on their **relationship** or **position** relative to each other in the HTML document tree. There are four main types of combinator selector

1. Descendant Selector (Space)

- **Description:** Selects an element that is a **descendant** of another element. A descendant can be an immediate child, a grandchild, or any element nested deeper within the specified ancestor.
- **Syntax:** The two selectors are separated by a **space** (S1 S2).
- **Example:** div p selects **all** <p> elements that are inside **any** element, no matter how deep the nesting.

2. Child Selector (>)

- **Description:** Selects an element that is a **direct child** of another element. This is a stricter relationship than the Descendant Selector.
- **Syntax:** The two selectors are separated by a **greater-than sign** (>).
- **Example:**ul > li selects only the **direct** elements whose immediate parent is a <ul> element. It would **not** select an <li> element nested inside a which is, in turn, inside the <ul>.

3. Adjacent Sibling Selector (+)

- **Description:** Selects an element that is the **immediate next sibling** of another element. Both elements must have the same parent, and the second element must come directly after the first in the source code.
- **Syntax:** The two selectors are separated by a **plus sign** (+).
- **Example:**
    - h1 + p selects the **first** <p> element that immediately follows an <h1> element and shares the same parent.

4. General Sibling Selector (~)

- **Description:** Selects an element that is a **sibling** of another element, regardless of whether it is immediate or not. Both elements must have the same parent, and the second element must appear *after* the first in the source code.
- **Syntax:** The two selectors are separated by a **tilde** (~).
- **Example:**
    - h2 ~ p selects **all** <p> elements that are siblings of an <h2> element and appear **after** it in the HTML, even if other elements are between them.

Program:

```
<html>
<head>
<title> Combinator Selectors example</title>
<style>
/*Descendent selector(space)*/
div p{
color:green;
font-size:16px;
}
/*Child selector(>)*/
div > ul {
background-color:#f0f0f0;
padding:10px;
}
/*Adjacent sibling selector(+)*/
h2 + p{
color:blue;
font-style:italic;
}
/*General sibling selector(~)*/
h3~ p{
color:red;
text-decoration:underline;
} </style>
</head>
<body>
<h1>Combinator Selectors Demo</h1>
<div> <p> This is a decsendant of div(styled using descendent selector)</p>
<section>
```

```
<p> This is nested deeper inside div(still in descendant)</p>
</section>
<ul> <li>Child of div-styled withchild selector</li>
<li>Another list item</li>
</ul> </div>
<h2>Heading 2</h2>
<p> This paragraph is an adjacent sibling of h2(styled using+selector)</p>
<p> this paragraph is not adjacnet to h2</p>
<h3>Heading 3</h2>
<p> this is general sibling of h3(styled using~selector)</p>
<p> Another general sibling of h3</p>
</body>
</html>
```

Output:

Combinator Selectors Demo

This is a decsendant of div(styled using descendent selector)

This is nested deeper inside div(still in descendant)

- Child of div-styled withchild selector
- Another list item

Heading 2

This paragraph is an adjacent sibling of h2(styled using+selector)

this paragraph is not adjacnet to h2

Heading 3

this is general sibling of h3(styled using~selector)

Another general sibling of h3

RESULT:

iii) Pseudo-class selector

Description:

A Pseudo-class selector in CSS is a keyword added to a selector that specifies a special state of the selected element(s). They allow you to select elements based on information that is not contained in the document tree (like its state, e.g., whether it has been visited), or to select a specific subset of elements from a group (e.g., the first child of a parent).

Pseudo-classes are always preceded by a colon (:).

They can generally be grouped into the following categories:

1. User Action / UI State Pseudo-classes

These classes style elements based on user interaction or their current state in the user interface.

Pseudo-class	Description	Example
:hover	Selects an element when the user's mouse pointer is over it.	a:hover
:active	Selects an element while it is being activated by the user (e.g., being clicked or pressed).	button:active
:focus	Selects an element that has input focus (e.g., when a user clicks into a text field).	input:focus
:checked	Selects radio buttons, checkboxes, or options in a <select> that are checked or toggled on.	input:checked
:disabled	Selects elements that are disabled (cannot be interacted with).	input:disabled

2. Link Pseudo-classes

These are used to style <a> (anchor) elements based on their visited status.

Pseudo-class	Description	Example
:link	Selects an unvisited link.	a:link
:visited	Selects a link that has been visited by the user.	a:visited

3. Structural Pseudo-classes (Positional Selectors)

These select elements based on their position relative to their siblings or parent.

Pseudo-class	Description	Example
:first−child	Selects an element that is the first child of its parent.	p:first-child

Pseudo-class	Description	Example
:last−child	Selects an element that is the last child of its parent.	li:last-child
:nth−child(n)	Selects an element based on its position among its siblings (where n can be a number, a keyword like *odd* or *even*, or a formula like 3n+1).	li:nth-child(odd)
:only−child	Selects an element that has no siblings (it is the only child of its parent).	p:only-child

Export to Sheets

4. Negation and Default Pseudo-classes

Pseudo-class	Description	Example
:not(selector)	Selects every element that is not specified by the selector inside the parentheses.	p:not(.intro)
:root	Selects the root element of the document (which is typically the <html> tag).	:root

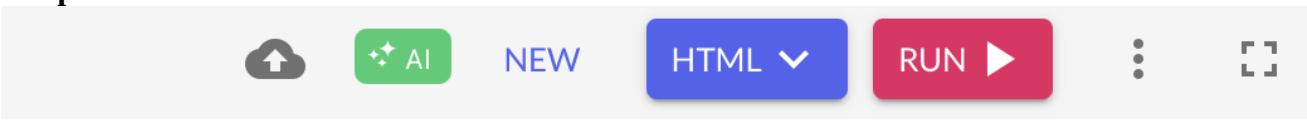
Program:

```
<html>
<head>
<title>Pseudo-Class Selectors Example</title>
<style>
/* :hover - when mouse is over the element */
a:hover {
  color: red;
  text-decoration: underline;
}
/* :first-child - targets the first child element */
ul li:first-child {
  font-weight: bold;
  color: green;
}
/* :last-child - targets the last child element */
```

```
ul li:last-child {
  font-style: italic;
  color: blue;
}
/* :nth-child - targets the nth child */
ul li:nth-child(2) {
  background-color: #ffffcc;
}
/* :focus - applies when input is focused */
input:focus {
  border: 2px solid orange;
  outline: none;
}
/* :checked - when checkbox is selected */
input[type="checkbox"]:checked + label {
color: darkblue;
  font-weight: bold;
}/* :visited - visited links */
a:visited {
  color: purple;
}
</style>
</head>
<body>
<h1>Pseudo-Class Selector Demo</h1>
<a href="https://www.google.com/" target="_blank">Hover or Visit me</a>
<form>
  <p>
    <label for="name">Name:</label>
    <input type="text" id="name" placeholder="Type your name">
```

```
  </p>
  <p>
    <input type="checkbox" id="agree">
    <label for="agree">I agree to terms</label>
  </p>
</form>
<!-- Example list to test first-child, last-child, nth-child -->
<ul>
  <li>First Item</li>
  <li>Second Item</li>
  <li>Last Item</li>
</ul>
</body>
</html>
```

Output:



Pseudo-Class Selector Demo

Hover or Visit me

Name: Yaswanth

I agree to terms

- **First Item**
- Second Item
- *Last Item*

Result:

iv) Pseudo-element selector

Description:

Based on the provided HTML and CSS code, a **Pseudo-element selector** is a CSS construct used to **style a specific part of an element** or to **insert content** before or after an element's content.

It is distinguished from a pseudo-class by using **double colons** (::) in modern CSS (though a single colon is still supported for backwards compatibility).

The code demonstrates the five most common pseudo-element selectors:

1. ::first−letter

- **Description:** Styles the **first letter** of the first line of a block-level element.
- This rule makes the very first letter of the text inside the <p> tag appear significantly larger, bold, and dark red.

2. ::first−line

- **Description:** Styles the **first formatted line** of a block-level element. The length of this line is dependent on the width of the element and the browser size.
- This rule makes the initial line of the text inside the <p> tag appear in green italics.

3. ::before

- **Description:** Inserts **generated content** immediately **before** the actual content of an element. This content is non-structural and often decorative, and it **requires** the content property.

This rule adds two orange asterisks (**) right before the text "CSS Magic" within the <h2> tag.

4. ::after

- **Description:** Inserts **generated content** immediately **after** the actual content of an element. Like ::before, it is non-structural and **requires** the content property.

This rule adds two orange asterisks (**) right after the text "CSS Magic" within the <h2> tag, completing the decorative border.

5. ::selection

- **Description:** Styles the portions of an element that have been **highlighted/selected** by a user (e.g., dragging the mouse over text).

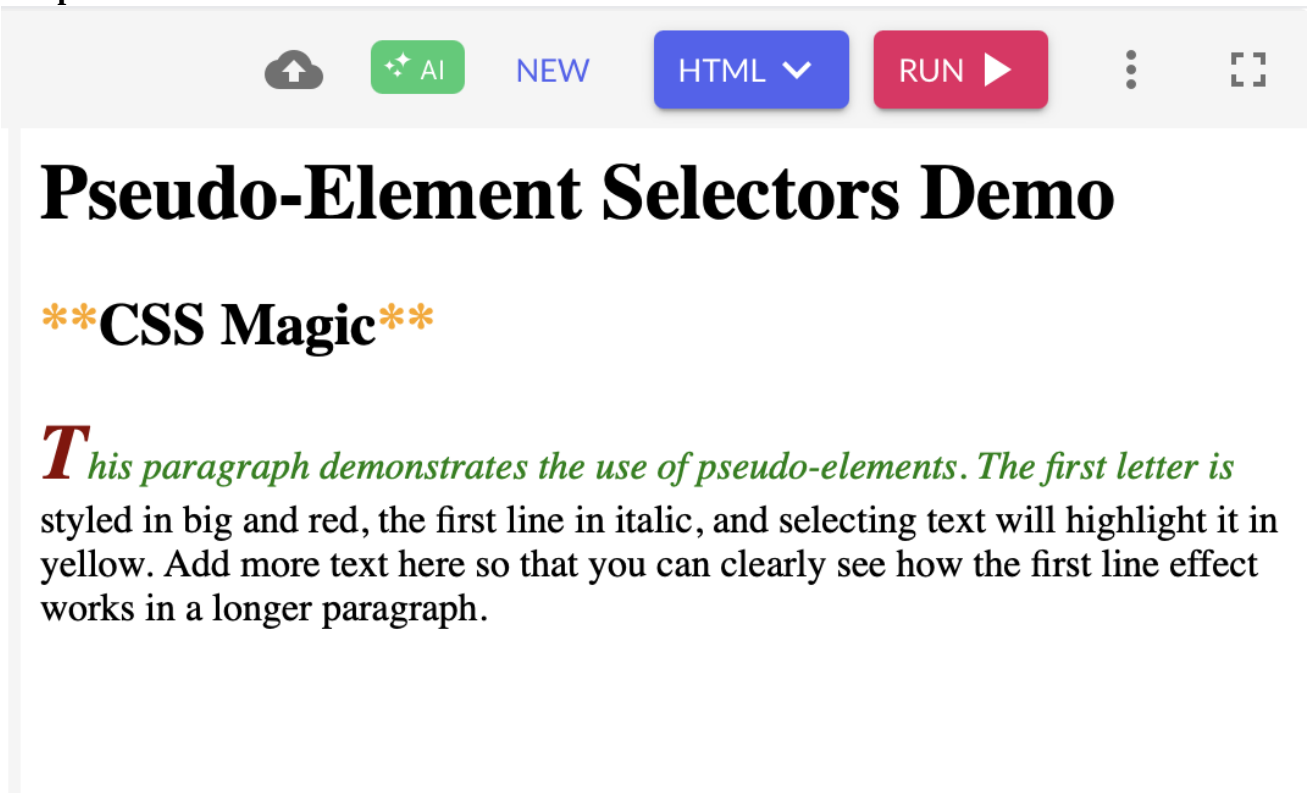
This universal rule changes the background of any selected text on the page to **yellow** and the text color to **black**.

Program:

```
<html>
<head>
<title>Pseudo-Element Selector Example</title>
<style>
/* ::first-letter - style the first letter of a paragraph */
p::first-letter {
  font-size: 200%;
  font-weight: bold;
  color: darkred;
}
/* ::first-line - style the first line of a paragraph */
p::first-line {
  font-style: italic;
  color: green;
}
/* ::before - insert content before an element */
h2::before {
  content: "**";
  color: orange;
}
/* ::after - insert content after an element */
h2::after {
  content: "**";
  color: orange;
}
/* ::selection - style selected text */
::selection {
  background-color: yellow;
  color: black;
```

```
}
</style>
</head>
<body>
<h1>Pseudo-Element Selectors Demo</h1>
<h2>CSS Magic</h2>
<p>
This paragraph demonstrates the use of pseudo-elements.
The first letter is styled in big and red,
the first line in italic, and selecting text will highlight it in yellow.
Add more text here so that you can clearly see how the first line effect works in a longer paragraph.
</p>
</body>
</html>
```

Output:



Result:

v) Attribute selector

Description:

Based on the provided code, an **Attribute Selector** in CSS is a way to select HTML elements based on the **presence of a specific attribute** or a **specific value** of an attribute.

This type of selector allows for highly targeted styling by looking at the key-value pairs within the HTML tags themselves, rather than relying only on the element name, ID, or class.

Key Characteristics and Examples from the Code

The code demonstrates three primary ways to use the Attribute Selector:

1. Selecting by Exact Attribute Value

This is the most common use, where you target elements that have an attribute set to a precise value.

- **Syntax:** element[attribute="value"]
- **Description:** This rule specifically selects and styles any <input> element whose type attribute is set **exactly** to "text".
- **Code Example:**

CSS

```
a[target="_blank"] {
  color: red;
}
```

- This rule selects and styles any <a> (anchor) element whose target attribute is set **exactly** to "_blank".

2. Selecting by Attribute Presence Only

This method targets any element that simply **has** the specified attribute, regardless of what its value is.

- **Syntax:** element[attribute]
- **Code Example:**

CSS

```
img[alt] {
  border: 2px solid green;
}
```

- This rule selects and styles any <img> element that **has** an alt attribute (which is used for alternative text), even though the value of alt is not explicitly checked.

Program:

```
<html>
<head>
<title>Simple Attribute Selector</title>
<style>
/* select input with type="text" */
input[type="text"] {
  background-color: lightyellow;
}
/* select link with target="_blank" */
a[target="_blank"] {
  color: red;
}
/* select image with alt attribute */
img[alt] {
  border: 2px solid green;
}
</style>
</head>
<body>
<h2>Attribute Selector Demo</h2>
<input type="text" placeholder="Enter your name"><br><br>
<a href="https://www.google.com/" target="_blank">Open in new Tab</a><br><br>
<!-- Better to use a relative path or hosted image -->
<img src="C:\Users\prasa\OneDrive\Pictures\download.jpeg" alt="sample image"
width="100" height="100">
</body>
</html>
```

Output:

Attribute Selector Demo

yaswanth

Open in new Tab

sample image

Result:

Expno:5a

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

Description:

Based on the provided code, colors in CSS can be referenced using multiple color models and notations, allowing developers to specify hues, saturation, lightness, and transparency with high precision.

The code demonstrates six common methods for defining the background-color of elements:

1. Named Colors

- Colors are specified using predefined, easily readable English names (keywords). These are the simplest method, though the range of named colors is limited compared to numeric models.
- Code Example:

CSS

```
.named-color{
  background-color:tomato; /* css named color */
}
```

2. Hexadecimal Notation (Hex Code)

- Colors are defined using a hash sign (#) followed by six hexadecimal characters (0-9 and A-F). The six digits are divided into three pairs, representing the amount of Red, Green, and Blue (RRGGBB), respectively.
- Code Example:

CSS

```
.hex-color{
  background-color:#1E90FF; /* dodger blue */
}
```

3. RGB Notation

- Colors are defined using the Red, Green, Blue color model. The rgb() function takes three comma-separated values, each ranging from 0 to 255, representing the intensity of the red, green, and blue components.
- Code Example:

CSS

```
.rgb-color{
  background-color:rgb(34,139,34); /* forest green */
```

}

4. RGBA Notation (with Transparency)

- This is an extension of the RGB model that includes an Alpha component. The rgba() function takes the standard Red, Green, and Blue values, plus a fourth value (the Alpha channel) that specifies the opacity or transparency.
- Alpha Value: The alpha value ranges from 0.0 (fully transparent) to 1.0 (fully opaque).
- Code Example:

CSS

```
.rgba-color{
  background-color:rgba(255,0,0,0.6); /* red with transparency */
}
```

5. HSL Notation

- Colors are defined using the Hue, Saturation, Lightness color model, which is often considered more intuitive than RGB.
    - Hue: The degree on the color wheel (0-360, where 0/360 is red, 120 is green, 240 is blue).
    - Saturation: The intensity of the color (0% is grayscale, 100% is full color).
    - Lightness: The brightness of the color (0% is black, 100% is white, 50% is "normal").
- Code Example:

CSS

```
.hsl-color{
  background-color:hsl(300,76%,72%); /* light purple */
}
```

6. HSLA Notation (with Transparency)

- This is an extension of the HSL model that includes an Alpha component for specifying transparency, similar to RGBA.
- Code Example:

CSS

```
.hsla-color{
  background-color:hsla(39,100%,50%,0.7); /* orange with transparency */
}
```

Program:

```
<html>
<head>
<title>CSS color Reference </title>
<style>
body{
font-family:Arial,sans-serif;
padding:20px;
background-color:#f9f9f9;
}
h1{
text-align:center;
}
.color-box{
width:250px;
height:100px;
color:white;
display:flex;
align-items:center;
justify-content:center;
margin:15px;
font-size:18px;
font-weight:bold;
border-radius:8px;
box-shadow:0 4px 6px rgba(0,0,0,0,1);
}
/*1.named color*/
.named-color{
background-color:tomata;/*css named color*/
}
```

```
/*2.hexa decimal notation*/
.hex-color{
background-color:#1E90FF;/*dodger blue*/
}
/*3.RGB notation*/
.rgb-color{
background-color:rgb(34,139,34);/*forest green*/
}
/*4.RGBA notation (with transperency)*/
.rgba-color{
background-color:rgba(255,0,0,0.6);/*red with transperency*/
}
/*5.HSL notation*/
.hsl-color{
background-color:hsl(300,76%,72%);/*light purple*/
}
/*6.HSLA notation(with transperency)*/
.hsla-color{
background-color:hsla(39,100%,50%,0.7);/*orange with transperency*/
}
</style>
</head>
<body>
<h1>different ways to reference colors in css(24B111CS171)</h1>
<div class="color-box named-color">Named color</div>
<div class="color-box hex-color">hex code</div>
<div class="color-box rgb-color">RGB</div>
<div class="color-box rgba-color">RGBA</div>
<div class="color-box hsl-color">HSL</div>
<div class="color-box hsla-color">HSLA</div>
```

</body>

</html>

Output:

Result:

Expno:5b

Aim: 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

Description:

Explanation (based on your program & the question)

1. position: fixed;
    - Locks the image relative to the viewport.
    - It does not move when scrolling up or down.
2. top: 50vh;
    - Places the image halfway down the viewport (vh = viewport height).
3. left: 50%; and transform: translate(-50%, -50%)
    - These two together perfectly center the image horizontally and vertically.
4. rotateZ(-12deg)
    - Rotates the image around the Z-axis → creates a horizontal tilt effect.
    - You can adjust angle for more/less tilt.
5. background-image + background-size: cover;
    - Makes sure the image fills the element without distortion.
6. Extra styling (border-radius, box-shadow, opacity)
    - For smoother corners, depth, and slight transparency.

In simple words

This CSS rule creates a tilted image box that always stays halfway down the screen no matter how much you scroll. The fixed positioning makes it “stick” in place, while rotation tilts it for a stylish look.

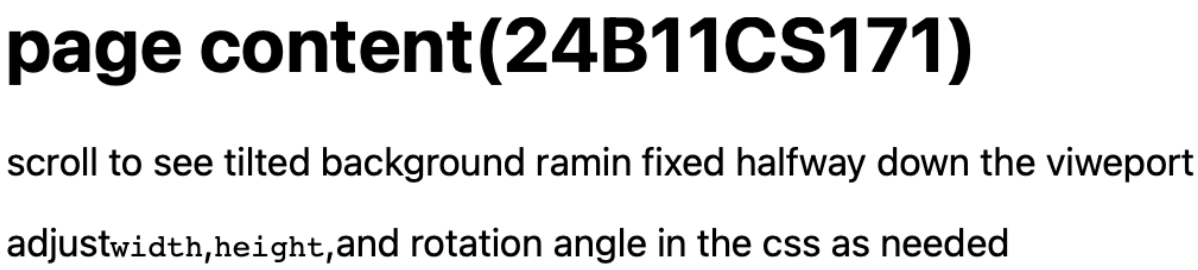
Program:

```
<html>
<head>
<title>fixed tilted background(horizontal rotate) </title>
<style>
/* container content so you can scroll the page*/
body{
```

```
min-height:300vh;/8make page scrollable to tset fixed behaviour*/
margin:0;
font-family:system-ui,Arial,sans-serif;
}
/* fixed background image element*/
.tilted-bg{
position:fixed;
left:50%;/* center horizontally*/
top:50vh;/* halfway down the view port*/
transform:translate(-50%,-50%) rotateZ(-12deg) translateZ(0);/* translate to center
,thenrotate; translateZ(0) triggers GPU*/
width:60vw;
height:40vh;
background-image:url("C:\Users\prasa\OneDrive\Pictures\Screenshots\Screenshot 2025-07-
13 120214.png");
background-size:cover;
background-position:center;
background-repeat:no-repeat;
pointer-events:none;/*so it wont block clicks*/
border-radius:12px;
box-shadow:0 10px 30px rgba(0,0,0,0.25);
z-index:9999;/* keep it on top(adjust if necessary)*/
opacity:0.95;
will-change:transform;
}
/*sample page content*/
main{
padding:2rem;
}
</style>
</head>
```

```
<body>
<div class="tilted-bg" aria-hidden="true"></div>
<main>
<h1>page content(24B11CS171)</h1>
<p>scroll to see tilted background ramin fixed halfway down the viweport</p>
<p>adjust<code>width</code>,<code>height</code>,and rotation angle in the css as needed</p>
<!--add content to scroll-->
<p style="margin-top:140vh">this line is roughly below the fold to show scrolling</p>
</main>
</body>
</html>
```

Output:



Result:

Expno:5c

Aim: 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

Description:

i. font-size

- Specifies the size of the font. This property directly controls how large or small the characters appear.
- Common Values**:** Can be set using absolute units (e.g., px - pixels), relative units (e.g., em or rem), percentages (e.g., 150%), or keywords (e.g., small, large).

ii. font-weight

- Sets the thickness or boldness of the characters in a text.
- Common Values:
    - Keywords: normal (equivalent to 400), bold (equivalent to 700).
    - Numeric values: Multiples of 100 from 100 (lightest) to 900 (heaviest/boldest). The availability of these exact numbers depends on the font family being used.

iii. font-style

- Specifies whether a font should be displayed in a normal, italic, or oblique style.
- Common Values:
    - normal: Displays the font normally.
    - italic: Displays the font in its italic face (if available in the font family).
    - oblique: Displays a synthetically slanted version of the font if an italic face is not available.

iv. text-decoration

- Used to set or remove decorations from text, such as underlines, overlines, or line-throughs. It is a shorthand property that can set text-decoration-line, text-decoration-color, text-decoration-style, and text-decoration-thickness.
- Common Values:
    - none: The default, removing all decorations (often used to style links).
    - underline: Draws a line beneath the text.
    - overline: Draws a line above the text.
    - line-through: Draws a line through the middle of the text.

v. text-transform

- Controls the capitalization of text, allowing you to convert text to uppercase, lowercase, or capitalize the first letter of each word, regardless of how it was entered in the HTML.
- Common Values:
    - none: Displays the text as it is.
    - uppercase: Converts all characters to capital letters.
    - lowercase: Converts all characters to small letters.
    - capitalize: Converts the first letter of each word to a capital letter.

vi. text-align

- Specifies the horizontal alignment of the inline content (text) within its block-level parent element.
- Common Values:
    - left: Aligns content to the left edge of the container.
    - right: Aligns content to the right edge of the container.
    - center: Centers the content horizontally.
    - justify: Stretches the lines of content so that the text aligns along both the left and right edges (common in newspaper columns).

Program:

```
<html>
<head>
<title> Css font and text properties</title>
<style>
body{
font-family: Arial, sans-serif;
padding:20px;
line-height:1.8;
}
/*1.font-size*/
.font-size{
font-size: 24px;/*increase text size*/
```

```
}
/*2.font-weight*/
.font-weight{
font-weight: bold;/*make text bold*/
}
/*3.font-style*/
.font-style{
font-style: italic;/*make text italic*/
}
/*4.text-decoration*/
.text-decoration{
text-decoration: underline;/*add underline*/
}
/*5.text-transform*/
.text-transform{
text-transform: uppercase;/*convert to uppercase*/
}
/*6.text-align*/
.text-align{
text-align:center;/*center align text*/
background-color:#f0f0f0;
padding:10px;
}
</style>
</head>
<body>
<h1>CSS FONT & TEXT PROPERTY DEMONSTARTION(24B111CS171)</h1>
<p class="font-size"> this text has a larger font size</p>
<p class="font-weight"> this text is bold</p>
<p class="font-style"> this text is italic</p>
```

 this text is underline

 this text is transformed to uppercase

 this text is centered

</body>

</html>

Output:

CSS FONT & TEXT PROPERTY DEMONSTARTION(24B11CS171)

this text has a larger font size

this text is bold

this text is italic

this text is underline

THIS TEXT IS TRANSFORMED TO UPPERCASE

this text is centered

Result:

Expno:5d

Aim: Write a program, to explain the importance of CSS Box model using

i.Content ii. Border iii. Margin iv. Padding

Description:

The CSS Box Model is a conceptual model used to calculate the size and spacing of every element rendered in a web browser. Every HTML element is treated as a rectangular box, composed of four distinct layers, moving from the inside out:

i. Content

- **Description:** This is the innermost area of the box. It contains the actual content of the element, such as text, images, or other embedded media.
- **Dimensions:** The width and height properties in CSS (e.g., width: 250px;) directly control the size of this Content area (by default, unless box-sizing: border-box; is used).
- **Purpose:** To display the data or media intended for the user.

ii. Padding

- **Description:** The space that surrounds the Content area and sits *inside* the Border.
- **Characteristics:** Padding is used to create visual space or "cushion" between the element's content and its border (or the edges of the box). It takes on the background color of the element itself.
- **Purpose:** To improve readability and visual separation within the element.

iii. Border

- **Description:** A structural line that wraps around the Padding and Content area.
- **Characteristics:** The Border is a configurable layer that can be given a thickness (border-width), a pattern (border-style), and a color (border-color).
- **Purpose:** To visibly define the edge of the element's box.

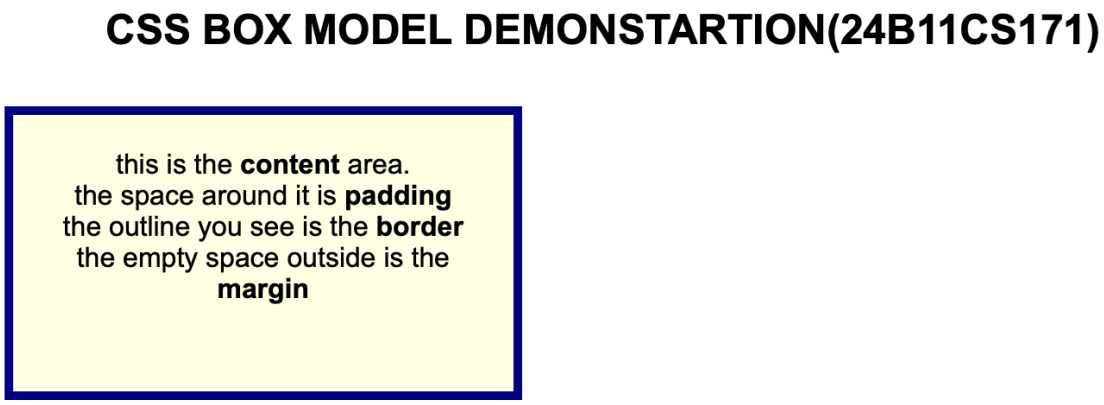
iv. Margin

- **Description:** The outermost layer that creates **space between the element and surrounding elements**. It sits *outside* the Border.
- **Characteristics:** The Margin area is always transparent and does not inherit the element's background color. Margins often collapse vertically, meaning the largest adjacent margin is used, not the sum of both.
- **Purpose:** To control the spacing and separation between different elements on the page.

Program:

```
<html>
<head>
<title>css box model</title>
<style>
.box{
width:250px;
height:120px;
background-color: lightyellow;/*content*/
padding:20px;/* padding*/
border: 5px solid darkblue;/*border*/
margin:30px;/8 margin*/
font-size: 16px;
text-align: center;
}
body{
backgroung-color:orange;
font-family:Arial,sans-serif;
}
</style
</head>
<body>
<h2 style="text-align:center">CSS BOX MODEL DEMONSTARTION(24B11CS171)</h2>
<div class="box">
this is the <b>content</b> area.<br>
the space around it is <b>padding</b><br>
the outline you see is the <b>border</b><br>
the empty space outside is the <b>margin</b></div>
</body>
</html>
```

Output:



Result:

Expno:6a

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

Description:

Internal JavaScript

- Written inside the HTML file using <script>...</script>.
- Useful for small scripts or quick demos.
- Example in program:
- <script>
- function showInternalMessage() {
- alert("Hello! This is INTERNAL JavaScript.");
- }
- </script>

2. External JavaScript

- Written in a separate .js file (like script.js).
- Linked into the HTML using <script src="filename.js"></script>.
- Useful for code reuse, maintainability, and cleaner HTML.
- Example in program:
- <script src="script.js"></script>

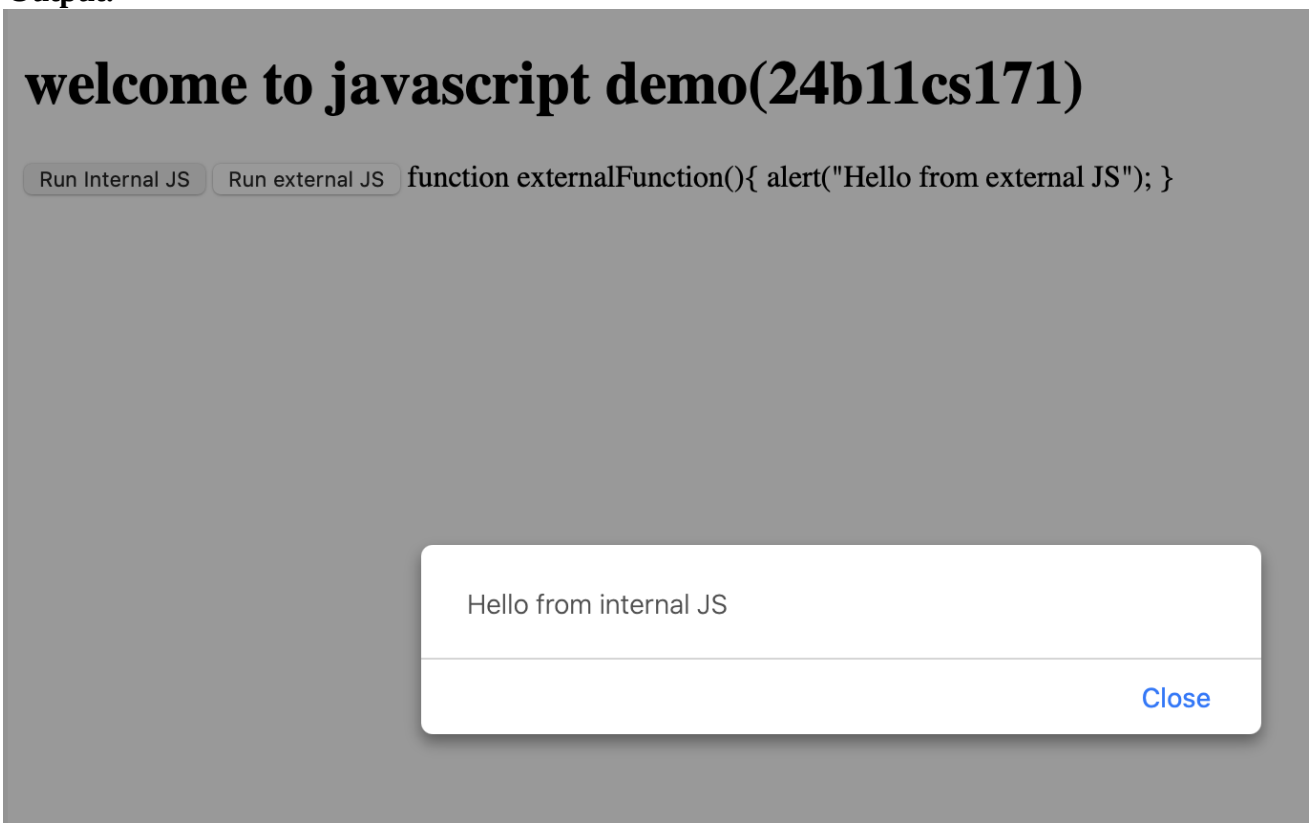
3. When to Use What

- Use internal JS for small, page-specific scripts.
- Use external JS for larger projects, multiple pages, or reusable functionality.

Program:

```
<!DOCTYPE html>
<html>
<head>
<title> java script embedding example</title>
<!--external javascript-->
<script src="script.js"></script>
</head>
<body>
<h1> welcome to javascript demo(24B11CS171)</h1>
<button onclick="internalFunction()">Run Internal JS</button>
<button onclick="externalFunction()">Run external JS</button>
<!--Internal Javascript-->
<script>
function internalFunction(){
alert("Hello from internal JS");
}
</script>
</body>
</html>
function externalFunction(){
alert("Hello from external JS");
}
```

Output:



Result:

Expno:6b

Aim: Write a program to explain the different ways for displaying output,

Description:

1. document (The DOM Interface)

- **Type:** A global object that represents the entire HTML page loaded in the browser window.
- **Purpose:** It serves as the entry point to the **Document Object Model (DOM)**. The DOM is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content.
- **Key Functions/Methods:**
    - **document.getElementById("id"):** Used to select a specific HTML element by its id attribute.
    - **$\text{document.querySelector(".class" or "#id")}$:** Used to select the first element that matches a specific CSS selector.
    - **document.createElement("tag"):** Used to dynamically create a new HTML element.
    - **document.write("text"):** Writes data directly to the HTML document stream (discouraged in modern web development, as it can overwrite existing content).
- **In Simple Terms:** If the HTML page is a house, the document object is the blueprint that allows you to find, examine, and change any wall, door, or window.

2. console (The Debugging Tool)

- **Type:** A global object that provides access to the browser's debugging console.
- **Purpose:** It is primarily used by developers to log non-visible output for debugging, tracking program flow, examining variables, and diagnosing errors.
- **Key Functions/Methods:**
    - **console.log("message"):** The most common method; outputs a general message to the console.
    - **console.error("message"):** Outputs an error message, usually styled with red text and a specific icon.
    - **console.warn("message"):** Outputs a warning message, often styled with yellow text.
    - **console.table(array/object):** Displays tabular data in a structured, readable format.

- **In Simple Terms:** The console is the developer's private notebook, allowing the program to whisper important information to the developer without showing it to the end user.

3. alert() (The User Communicator)

- **Type:** A global function of the window object (called directly as alert()).
- **Purpose:** It displays a simple, modal dialog box (a small pop-up window) with a message and an "OK" button to the user.
- **Key Characteristics:**
    - **Modal:** The alert box is modal, meaning it **pauses the execution of the entire page script** until the user dismisses the box by clicking "OK."
    - **Basic UI:** The appearance of the alert box cannot be customized; its style is dictated by the user's operating system or browser.
- **Usage Note:** Due to its disruptive, unstylable nature, the use of alert() is generally discouraged in modern production applications, where custom, non-blocking notification systems are preferred.
- **In Simple Terms:** alert() is like an emergency megaphone for the JavaScript code, halting everything to deliver a critical (or just quick) message to the user.

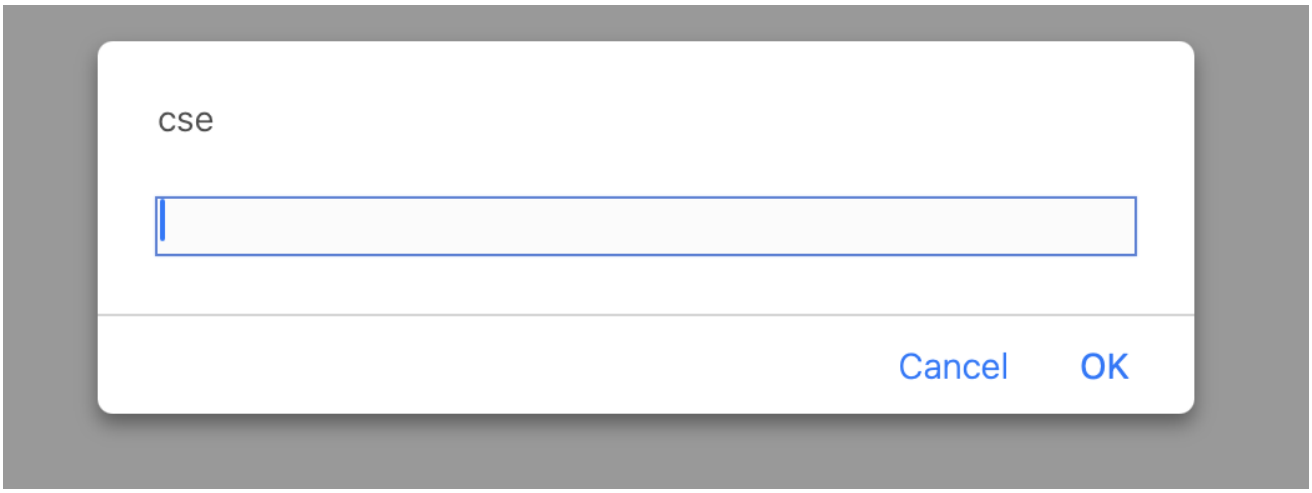
Program:

```
<!DOCTYPE html>
<html>
<head>
<title> multiple print</title>
</head>
<body>
<h1> multiple print</h1>
<script>
document.write("sagar");
console.log("135");
prompt("cse");
</script>
</body>
</html>
```

Output:

multiple print

YASWANTH



Result:

Expno:6c

Aim: Write a program to explain different ways for taking input

Description:

Text Input Field (<input type="text">)

- Users can type text in a field.
- Accessed using document.getElementById("id").value.
- Example: Name input in the program.

Prompt Box (prompt())

- A pop-up dialog asking the user for input.
- Returns the entered value as a string.
- Example: Age input in the program.

Form Submission (<form> + submit button)

- Users submit inputs using a form.
- Use onsubmit and event.preventDefault() to capture data without reloading the page.
- Example: Email input in the program.

Dropdown Selection (<select>)

- Users select from predefined options.
- Access selected value using .value.
- Example: Country selection in the program.

How It Works

- All inputs are displayed dynamically in the .
- Each method demonstrates a **different way JavaScript interacts with user input**.
- This program is perfect for beginners to **learn how to capture and use input values**.

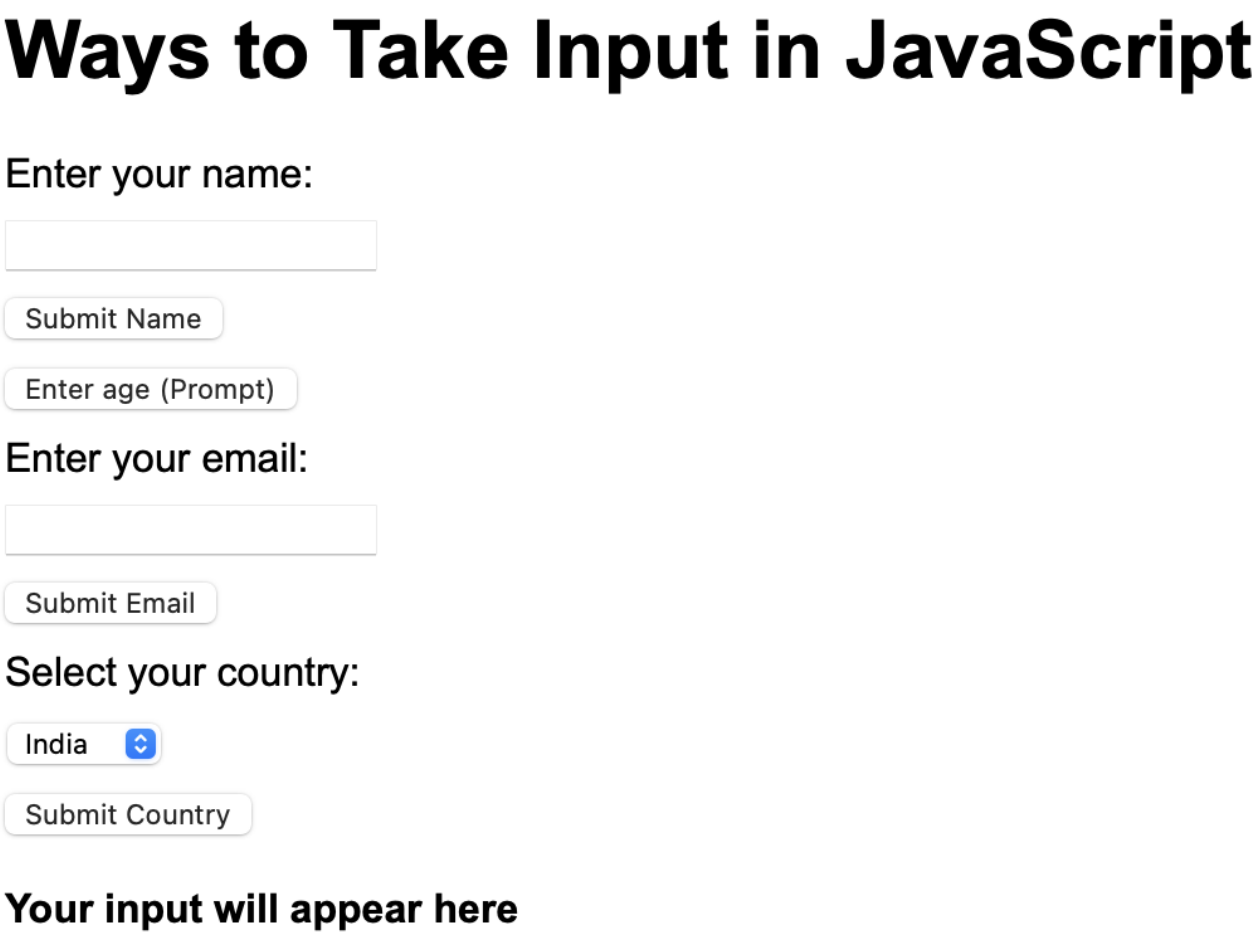
Program:

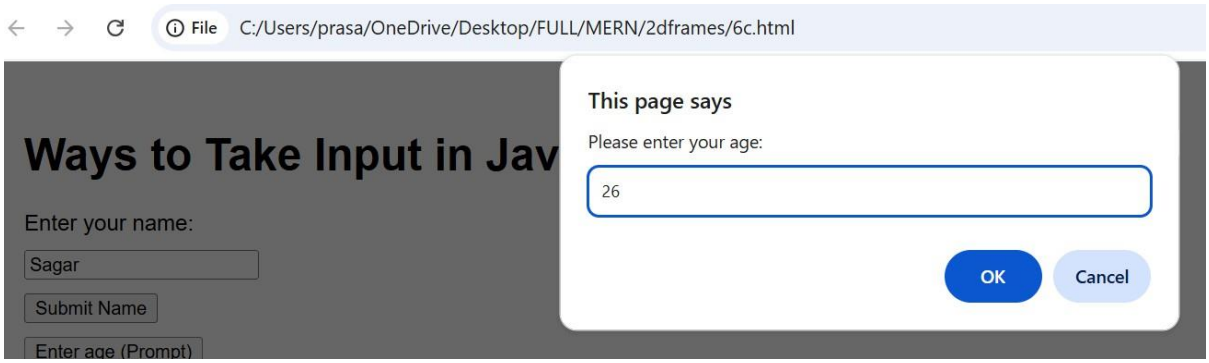
```
<!DOCTYPE html>
<html>
<head>
<title>JS Input Methods</title>
<style>
body {
  font-family: Arial, sans-serif;
  padding: 20px;
}
input, button, select {
  margin: 10px 0;
  display: block;
}
#output {
  margin-top: 20px;
  font-weight: bold;
}
</style>
</head>
<body>
<h1>Ways to Take Input in JavaScript</h1>
   <!-- 1. Text input field -->
<label>Enter your name:</label>
<input type="text" id="nameInput">
<button onclick="getInputFromField()">Submit Name</button>
<!-- 2. Prompt Box -->
<button onclick="getInputFromPrompt()">Enter age (Prompt)</button>
<!-- 3. Form input -->
<form onsubmit="getInputFromForm(event)">
```

```
  <label>Enter your email:</label>
  <input type="email" id="emailInput">
  <button type="submit">Submit Email</button>
</form>
<!-- 4. Dropdown selection -->
<label>Select your country:</label>
<select id="countrySelect">
  <option value="India">India</option>
  <option value="USA">USA</option>
  <option value="Japan">Japan</option>
</select>
<button onclick="getInputFromDropdown()">Submit Country</button>
<div id="output">Your input will appear here</div>
<script>
// 1. Input from text field
function getInputFromField() {
  const name = document.getElementById("nameInput").value;
  document.getElementById("output").innerText = "Name: " + name;
}
// 2. Input from prompt box
function getInputFromPrompt() {
  const age = prompt("Please enter your age:");
  document.getElementById("output").innerText = "Age: " + age;
}
// 3. Input from form submission
function getInputFromForm(event) {
  event.preventDefault(); // Prevent page reload
  const email = document.getElementById("emailInput").value;
  document.getElementById("output").innerText = "Email: " + email;
}
```

```
// 4. Input from dropdown selection
function getInputFromDropdown() {
  const country = document.getElementById("countrySelect").value;
  document.getElementById("output").innerText = "Country: " + country;
}
</script>
</body>
</html>
```

Output:





Ways to Take Input in JavaScript

Enter your name:

Yash

Submit Name

Enter age (Prompt)

Enter your email:

yash@gmail.com

Submit Email

Select your country:

India

Submit Country

Your input will appear here

Result:

Expno:6d

Aim: 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

Description:

Function: checkVoter()

- Prompt input:
- const name = prompt("Enter your name");
- const age = prompt("Enter your age");
    - Uses prompt dialog boxes to collect user input for name and age.
- Input validation:
- if (name === null || age === null || name.trim() === "" || isNaN(age.trim())) {
- alert("Invalid input. Please try again");
- return;
- }
    - Checks if the user canceled the prompt or entered invalid data.
    - trim() removes extra spaces.
    - isNaN() ensures age is numeric.
- Negative age check:
- if (ageNum < 0) {
- alert("Age cannot be negative");
- return;
- }
- Eligibility determination:
- const eligibility = ageNum >= 18 ? "Eligible to vote" : "Not eligible to vote";
    - Uses ternary operator to decide if voter is eligible based on age ≥ 18.
    - Constructs HTML table as a string with the input and eligibility status.
    - Uses template literals (${variable}) for inserting values.

  document.getElementById("result").innerHTML = tableHTML;
    - Inserts the table into the page dynamically.

4. How It Works

1. User clicks “Check Eligibility” button.

2. Prompt dialogs appear asking for name and age.
3. Input is validated for emptiness, numeric value, and negativity.
4. Age is checked for eligibility (≥18 = eligible).
5. A table is generated dynamically showing:
    - Name
    - Age
    - Eligibility status
6. The table is displayed in the .

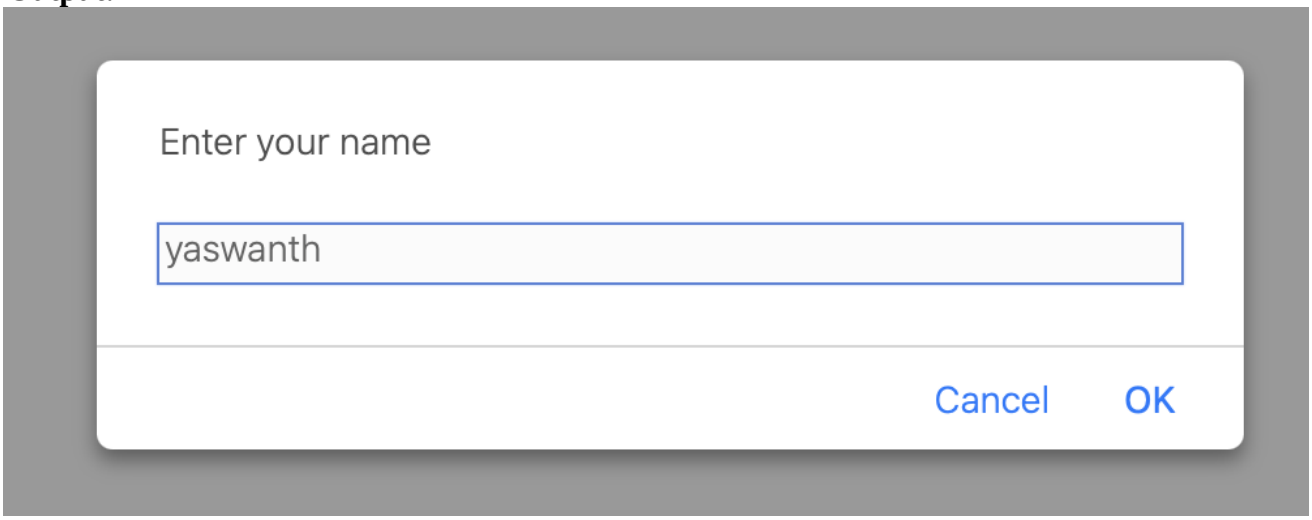
Program:

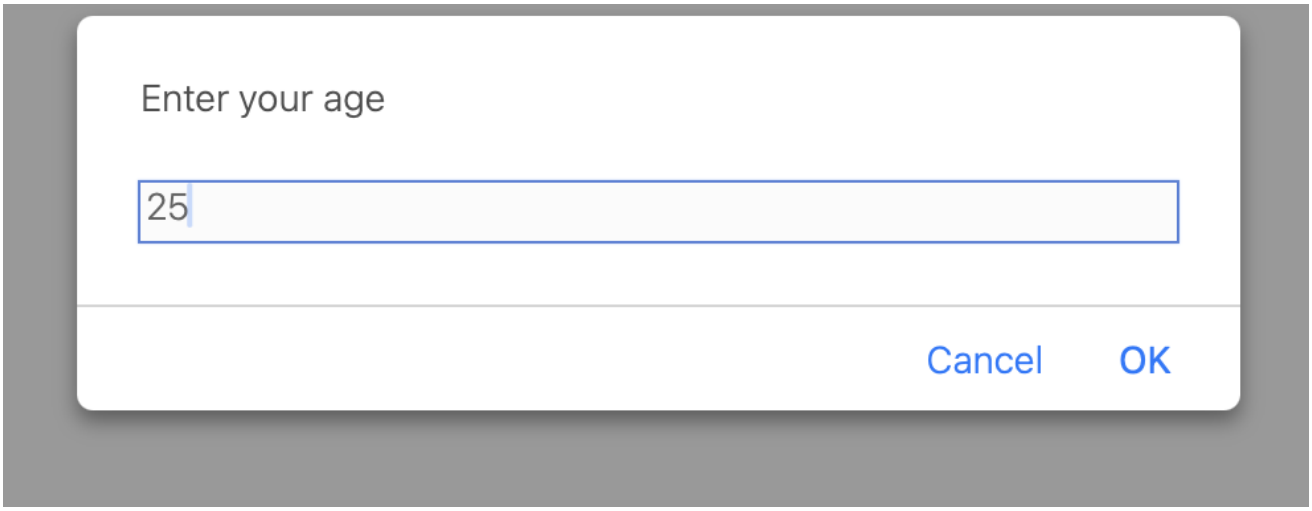
```
<html>
<head>
  <title>Voter Eligibility Check</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
    }
    table {
      border-collapse: collapse;
      width: 50%;
      margin-top: 20px;
    }
    th, td {
      border: 1px solid #333;
      padding: 10px;
      text-align: center;
    }
    th {
      background-color: tomato;
```

```
            color: white;
        }
    </style>
</head>
<body>
    <h1>Voter Eligibility Checker</h1>
    <button onclick="checkVoter()">Check Eligibility</button>
    <div id="result"></div>
  <script>
        function checkVoter() {
            const name = prompt("Enter your name");
            const age = prompt("Enter your age");

            if (name === null || age === null || name.trim() === "" || isNaN(age.trim())) {
                alert("Invalid input. Please try again");
                return;
            }

            const ageNum = parseInt(age.trim());

            if (ageNum < 0) {
                alert("Age cannot be negative");
                return;
            }

            const eligibility = ageNum >= 18 ? "Eligible to vote" : "Not eligible to vote";

            const tableHTML = `
                <table>
                    <tr>
```

```
                <th>Name</th>
                <th>Age</th>
                <th>Eligibility</th>
            </tr>
            <tr>
                <td>${name}</td>
                <td>${ageNum}</td>
                <td>${eligibility}</td>
            </tr>
        </table>
    `;
  document.getElementById("result").innerHTML = tableHTML;
    }
  </script>
</body>
</html>
```

Output:





Voter Eligibility Checker

Check Eligibility

Name	Age	Eligibility
yaswanth	25	Eligible to vote

Voter Eligibility Checker

Check Eligibility

Name	Age	Eligibility
yash	14	Not eligible to vote

Result:

Expno:7a

Aim: Write a program using document object properties and methods

Description:

JavaScript Functionality (showInfo() function):

- Document Properties Used:
  - document.title → Retrieves the title of the web page.
  - document.url → Retrieves the URL of the current page.
  - document.lastModified → Retrieves the date and time when the document was last modified.
- Document Methods Used:
  - document.getElementById("heading").style.color="blue" → Changes the color of the heading text to blue.
  - document.getElementById("output").innerHTML → Displays the document information (title, URL, last modified date) inside the output .

How It Works:

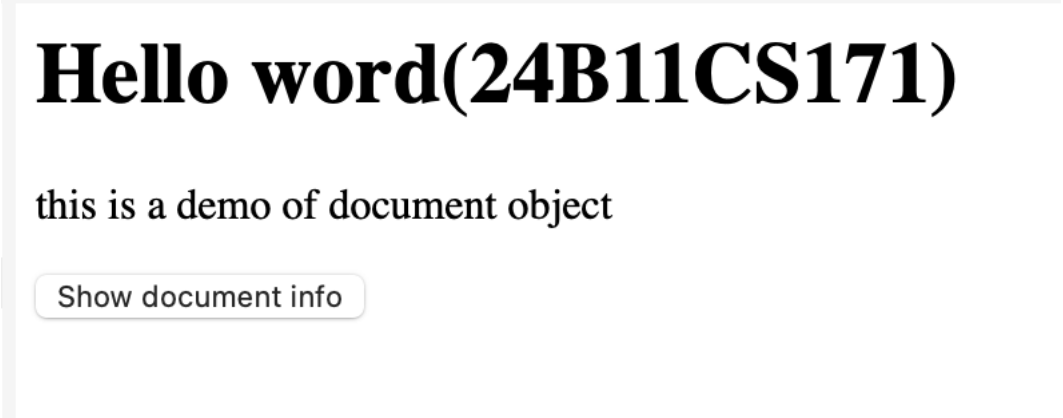
- When the user clicks the "Show document info" button, the showInfo() function is called.
- The function retrieves document properties and updates the HTML content dynamically.
- The heading color is changed to visually indicate that the function has executed successfully.
- All document information is displayed neatly in the .

Program:

```
<html>
<head>
<title>document object example</title>
</head>
<body>
<h1 id="heading">Hello word(24B11CS171)</h1>
<p>this is a demo of document object</p>
<button onclick="showInfo()">Show document info</button>
```

```
<div id="output"></div>
<script>
function showInfo(){
//using document properties
let title=document.title;
let url=document.url;
let lastModified=document.lastModified;
//using documnet methods
document.getElementById("heading").style.color="blue";//change heading color
document.getElementById("output").inner.Html=
"Title: "+ title + "<br>"+
"URL: "+ url +"<br>"+
"Last Modified:" + lastModified;
}
</script>
</body>
</html>
```

Output:



Result:

Expno:7b

Aim: Write a program using window object properties and methods

Description:

a) showWindowInfo() function

- Uses window object properties:
  - window.innerWidth → Gets the width of the browser window's content area.
  - window.innerHeight → Gets the height of the browser window's content area.
  - window.location.href → Retrieves the current URL of the page.
- Updates the content of the using document.getElementById("output").innerHTML to display the window information.

b) openNewWindow() function

- Uses window.open() method to open a new window with:
  - URL: "https://www.google.com"
  - Name: "Example"
  - Size: width=400, height=300
- Stores the reference of the new window in the variable newWin for later use.

c) closeNewWindow() function

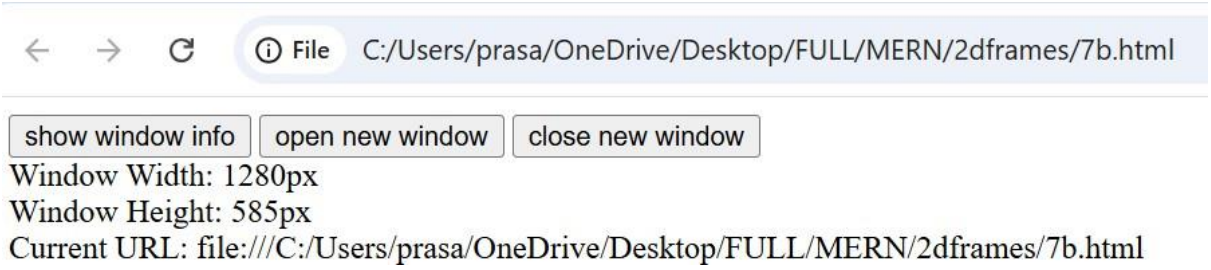
- Uses window.close() method to close the newly opened window.
- Checks if the window reference newWin exists before attempting to close it.

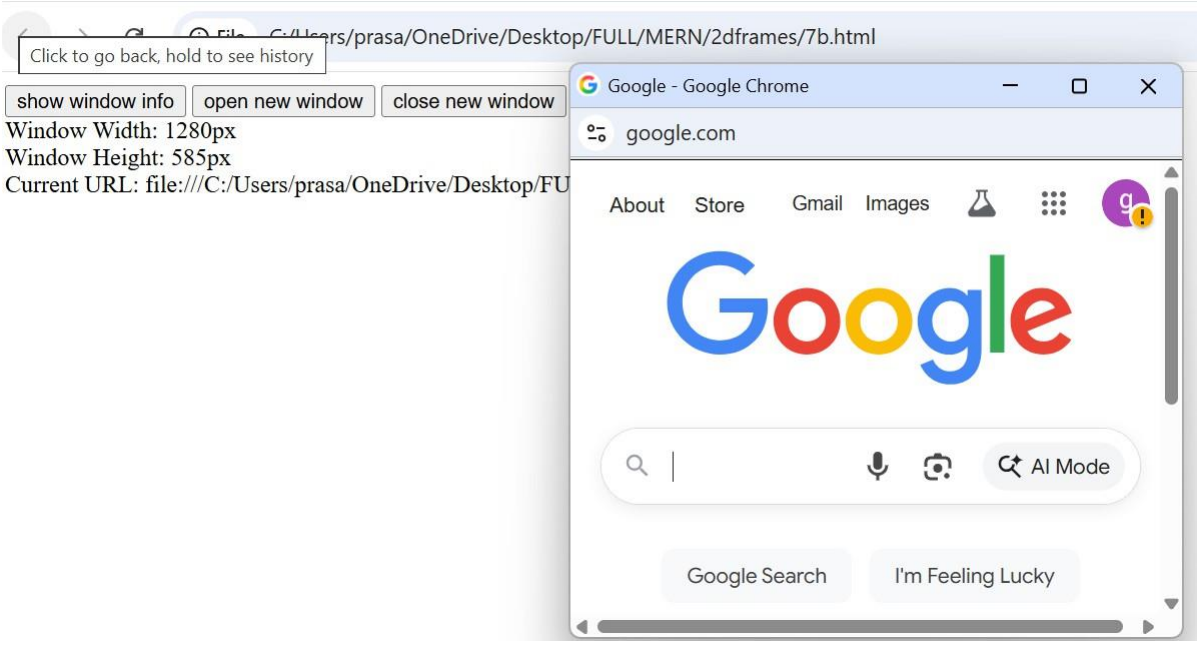
Program:

```
<html>
<head>
<title>window object example</title>
</head>
<body>
<button onclick="showWindowInfo()">show window info</button>
<button onclick="openNewWindow()">open new  window </button>
<button onclick="closeNewWindow()">close new window</button>
<div id="output"></div>
<script>
let newWin;//to store reference of opened window
function showWindowInfo(){
//using window properties
let width=window.innerWidth;
let height=window.innerHeight;
let locationHref=window.location.href;
//display info
document.getElementById("output").innerHTML=
"Window Width: " + width + "px<br>" +
"Window Height: " + height + "px<br>" +
"Current URL: " + locationHref ;
}
function openNewWindow(){
//using window.open()
newWin = window.open("https://www.google.com", "Example","width=400, height=300");
}
function closeNewWindow(){
//using window.close
if(newWin){
```

```
newWin.close();
}
}
</script>
</body>
</html>
```

Output:





Result:

Expno:7c

Aim: Write a program using array object properties and methods

Description:

Array Creation

- An array named fruits is created with three elements:
- let fruits = ["Apple", "Banana", "Mango"];

Array Property

- fruits.length → Retrieves the total number of elements in the array (in this case, 3).

Array Methods

- push("Orange") → Adds "Orange" to the end of the array.
- unshift("Grapes") → Adds "Grapes" to the beginning of the array.
- pop() → Removes the last element of the array and returns it.
- shift() → Removes the first element of the array and returns it.
- sort() → Sorts the array elements alphabetically.
- join(",") → Combines all elements into a single string, separated by comma

Program:

```
<html>
<head>
<title>Array object properties</title>
</head>
<body>
<h2>Array object example</h2>
<p id="output"></p>
<script>
//creating an array
let fruits = ["Apple", "Banana", "Mango"];
//array property: length
let length = fruits.length;
//array methods
```

```
fruits.push("Orange");        // adds element at end
fruits.unshift("Grapes");     // adds element at beginning
let removedLast = fruits.pop();  // removes last element
let removedFirst = fruits.shift(); // removes first element
let sorted = fruits.sort();  // sorts the array
let joined = fruits.join(","); // joins elements as string
//display results
document.getElementById("output").innerHTML =
"<b>Original length:</b> " + length + "<br>" +
"<b>After push & unshift:</b> " + fruits.join(",") + "<br>" +
"<b>Removed last:</b> " + removedLast + "<br>" +
"<b>Removed first:</b> " + removedFirst + "<br>" +
"<b>Sorted Array:</b> " + sorted + "<br>" +
"<b>Joined as string:</b> " + joined;
</script>
</body>
</html>
```

Output:

Result:

Expno:7d

Aim: Write a program using math object properties and methods

Description:

Math Object Properties Used

- Math.PI → Returns the value of π (approximately 3.14159).
- Math.E → Returns Euler's number (2.718), the base of natural logarithms.
- Math.SQRT2 → Returns the square root of 2 (approximately 1.414).

Math Object Methods Used

- Math.sqrt(16) → Returns the square root (4).
- Math.pow(2,5) → Returns 2 raised to the power of 5 (32).
- Math.abs(-25) → Returns the absolute value (25).
- Math.round(4.7) → Rounds to the nearest integer (5).
- Math.ceil(4.2) → Rounds up to the next integer (5).
- Math.floor(4.8) → Rounds down to the nearest integer (4).
- Math.random() → Returns a random number between 0 and 1 (varies each time).
- Math.max(3,9,4,6,98) → Returns the maximum value (98).
- Math.min(3,9,4,6,98) → Returns the minimum value (3).

Program:

```
<html>
<head>
<body>
<h2>Math object example</h2>
<p id="output"></p>
<script>
//math object properties
let piValue=Math.PI;  //value of pi
let eValue=Math.E;  //value of eulers number
let sqrt2Value=Math.SQRT2; //square root of 2
//math object methods
let sqrtResult=Math.sqrt(16); //square root
let powerResult=Math.pow(2,5); //2^5
let absResult=Math.abs(-25); //absolute value
let roundResult=Math.round(4.7);
let ceilResult=Math.ceil(4.2);
let floorResult=Math.floor(4.8);
let randomResult=Math.random();
let maxResult=Math.max(3,9,4,6,98);
let minResult=Math.min(3,9,4,6,98);
//display result
document.getElementById("output").innerHTML =
"<b>Properties:</b><br>" +
"PI: " +piValue +"<br>" +
"E: " +eValue + "<br>" +
"SORT2: " + sqrt2Value + "<br><br>" +
"<b>Methods:</b><br>"+
"Square Root of 16:" + sqrtResult + "<br>" +
"2^5 (Power): " +powerResult + "<br>" +
```

```
"Absolute of -25: " + absResult +"<br>" +
"Round(4.7): " + roundResult +"<br>" +
"Ceil(4.2): " + ceilResult +"<br>" +
"Floor(4.8): " + floorResult +"<br>" +
"Random Number(0-1): " + randomResult +"<br>" +
"Max(3,9,4,6,98) " + maxResult +"<br>" +
"Min(3,9,4,6,98) " + minResult;
</script>
</body>
</html>
```

Output:

Math object example

Properties:
PI: 3.141592653589793
E: 2.718281828459045
SORT2: 1.4142135623730951

Methods:
Square Root of 16:4
2^5 (Power): 32
Absolute of -25: 25
Round(4.7): 5
Ceil(4.2): 5
Floor(4.8): 4
Random Number(0-1): 0.9912296708570519
Max(3,9,4,6,98) 98
Min(3,9,4,6,98) 3

Result:

Expno:7e

Aim: Write a program using string object properties and methods

Description:

1. String Creation
    - A string is created:
    - let text = "Hello Javascript World!";
2. String Property Used
    - text.length → Returns the total number of characters in the string (23 in this case, including spaces).
3. String Methods Used
    - toUpperCase() → Converts all characters to uppercase.
      → "HELLO JAVASCRIPT WORLD!"
    - toLowerCase() → Converts all characters to lowercase.
      → "hello javascript world!"
    - trim() → Removes leading and trailing spaces (not visible here since the string has no extra spaces).
    - slice(6,16) → Extracts part of the string from index 6 up to (but not including) 16.
      → "Javascript"
    - replace("World","programmers") → Replaces the first occurrence of "World" with "programmers".
      → "Hello Javascript programmers!"
    - charAt(7) → Returns the character at index 7.
      → "a"
    - indexOf("Javascript") → Returns the position (index) where "Javascript" starts.
      → 6
    - split(" ") → Splits the string into an array of words based on spaces.
      → ["Hello","Javascript","World!"]
4. Displaying Results
    - All results are shown inside a <p> element with id="output" using innerHTML.

Program:

```
<html>
<head>
<title>string object exapmle</title>
</head>
<body>
<h2> javascript String object example</h2>
<p id="output"></p>
<script>
//creating an string
let text="Hello Javascript World!";
//using an stringproperties
let length=text.length;//property
//using an string methods
let  upper=text.toUpperCase();
let lower=text.toLowerCase();
let trimmed=text.trim();
let sliced=text.slice(6,16);
let replaced=text.replace("World","programmers");
let charAtPos=text.charAt(7);
let index=text.indexOf("Javascript");
let splitted=text.trim().split(" ");
//display results
document.getElementById("output").innerHTML =
"<b>Original String:</b> " + text + "<br>" +
"<b>Length:</b> " + length + "<br>" +
"<b>Uppercase:</b> " + upper + "<br>" +
"<b>Lowercase:</b> " + lower + "<br>" +
"<b>Trimmed:</b> " + trimmed + "<br>" +
"<b>Sliced(6,16):</b> " + sliced + "<br>" +
```

```
"<b>Replaced:</b> " + replaced + "<br>" +
"<b>Character at 7:</b> " + charAtPos + "<br>" +
"<b>Index of 'JavaScript':</b> " + index + "<br>" +
"<b>Splitted:</b> " + splitted.join(",");
</script>
</body>
</html>
```

Output:

Result:

Expno:7g

Aim: Write a program using date object properties and methods

Description:

Creating a Date Object

let CurrentDate = new Date();

- This creates a new Date object containing the current system date and time.

Date Object Methods Used

- getFullYear() → Returns the **year** (e.g., 2025).
- getMonth() → Returns the **month** (0–11). Since January is 0, we add +1 for readability.
- getDate() → Returns the **day of the month** (1–31).
- getDay() → Returns the **day of the week** (0–6, where 0 = Sunday).
- getHours() → Returns the **current hour** (0–23).
- getMinutes() → Returns the **minutes** (0–59).
- getSeconds() → Returns the **seconds** (0–59).
- getTime() → Returns the number of **milliseconds since January 1, 1970** (Unix epoch).

Program:

<html>

<head>

<title>Date object exapmle</title>

</head>

<body>

<h2> javascript Date object example</h2>

<script>

//create a new date object

let CurrentDate = new Date();

//use date object methods

```
let year=CurrentDate.getFullYear();
let month=CurrentDate.getMonth() + 1;
let date=CurrentDate.getDate();
let day=CurrentDate.getDay();
let hours=CurrentDate.getHours();
let minutes=CurrentDate.getMinutes();
let seconds=CurrentDate.getSeconds();
let time=CurrentDate.getTime();
//Array for week days
let days=["sunday","monday","tuesday","wednesday","thursday","friday","saturday"];
//display the values
document.getElementById("dateInfo").innerHTML =
"Current Date: " + date + "/" + month + "/" + year + "<br>" +
"Day of the week: " + days[day] + "<br>" +
"Current Time : " + hours + ":" + minutes + ":" + seconds + "<br>" +
"Milliseconds since Jan 1, 1970: " + time;
</script>
</body>
</html>
```

Output:

Result:

Expno:7h

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

Description:

Constructor Function

function Student(name, age, marks) { ... }

- A **constructor function** Student is defined to create multiple student objects.
- It initializes each student with **properties**: name, age, and marks.

Properties

- this.name → Stores the student's name.
- this.age → Stores the student's age.
- this.marks → Stores the student's marks.

Method

- displayDetails() → Returns a formatted string with the student's name, age, and marks.
  Example:Name: Ram, Age: 20, Marks: 92

Accessors (Getters & Setters)

- **Getter → getGrade()**
  - Determines the grade based on marks:
    - >=90 → Grade A
    - >=75 → Grade B
    - >=50 → Grade C
    - <50 → Fail
- **Setter → setMarks(newMarks)**
  - Updates the student's marks to a new value.

Creating and Using an Object

- let student1 = new Student("Ram", 20, 85);
  Creates an object student1 with initial values.
- student1.setMarks(92);
  Updates the marks using the **setter method**.
- student1.displayDetails() and student1.getGrade() are called to display details and grade.

Program:

```
<html>
<head>
<title>user-defined object exapmle</title>
</head>
<body>
<h2> User-Defined object in javascript(24B11CS171)</h2>
<p id="output"></p>
<script>
//constructor function to define a user-defined object
function Student(name,age,marks) {
//properties
this.name=name;
this.age=age;
this.marks=marks;
//method
this.displayDetails=function() {
return "Name: " + this.name + ", Age: " + this.age + ",Marks: " + this.marks;
};
//Accessor(Getter)
this.getGrade=function(){
if(this.marks>=90) return "A";
else if(this.marks>=75) return "B";
else if(this.marks>=50) return "C";
else return "Fail";
};

//Accessor(Setter)
this.setMarks=function(newMarks){
this.marks=newMarks;
```

```
};
}
//create object using constructor
let student1=new Student("Ram",20,85);
//use setter to update marks
student1.setMarks(92);
//Display details and grade
document.getElementById("output").innerHTML=
student1.displayDetails()+ "<br>" +
"Grade: " + student1.getGrade();
</script>
</body>
</html>
```

Output:

User-Defined object in javascript(24B11CS171)

Result:

Expno:7f

AIM: Write a program using validation object properties and methods

DESCRIPTION:

- The page is titled “Contact Us Page” and contains a form for user details.
- Includes input fields for Name, Email, Password, Phone Number, and Address.
- Uses a JavaScript function validation() to check user inputs before submission.
- Ensures all fields are not empty before allowing the form to submit.
- Validates name length (minimum 4 characters).
- Checks email format using a regular expression pattern.
- Displays alert messages for missing or invalid inputs.
- Uses HTML <form> with onSubmit="return validation()" to trigger validation.
- If all validations pass, data is submitted to registration_app.php.
- Provides Submit and Reset buttons for user convenience.

PROGRAM:

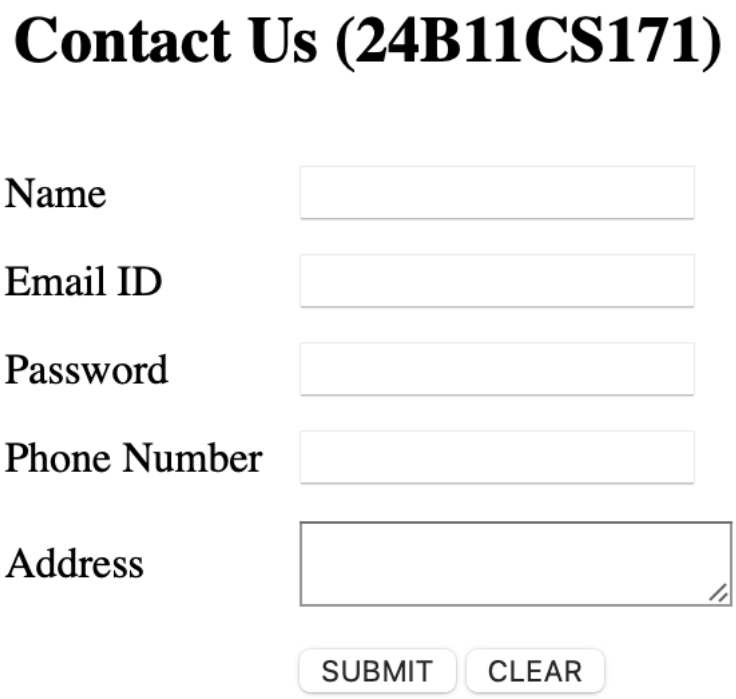
```
<html>
<head>
   <title>Contact Us Page</title>
   <script type="text/javascript">
      function validation() {
         var nm = document.getElementById("name_id").value.trim();
         var em = document.getElementById("email_id").value.trim();
         var pw = document.getElementById("password_id").value.trim();
         var ph = document.getElementById("phone").value.trim();
         var ad = document.getElementById("address").value.trim();

         if (nm === "" || em === "" || pw === "" || ph === "" || ad === "") {
            alert("Please fill in all fields");
            return false;
```

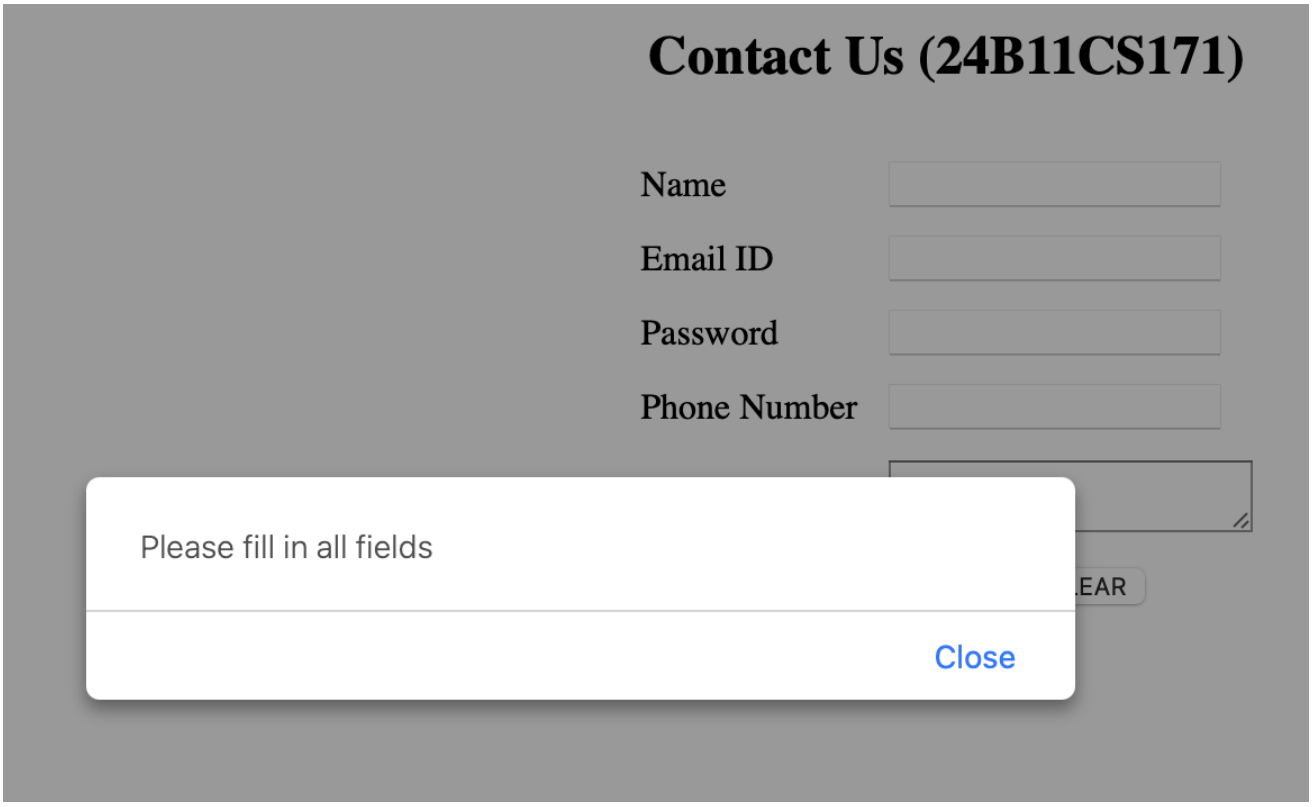
```
        }
        else if (nm.length < 4) {
            alert("Name should be at least 4 characters long");
            return false;
        }
        else if (!/^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$/.test(em)) {
            alert("Enter a valid Email ID");
            return false;
        }
        else if (pw.length < 6) {
            alert("Password should be at least 6 characters long");
            return false;
        }
        else if (!/^\d{10}$/.test(ph)) {
            alert("Enter a valid 10-digit phone number");
            return false;
        }
        else if (ad.length < 20) {
            alert("Address should be at least 20 characters long");
            return false;
        }
        else {
            alert("Form submitted successfully!");
            return true;
        }
    }
  </script>
</head>

<body>
```

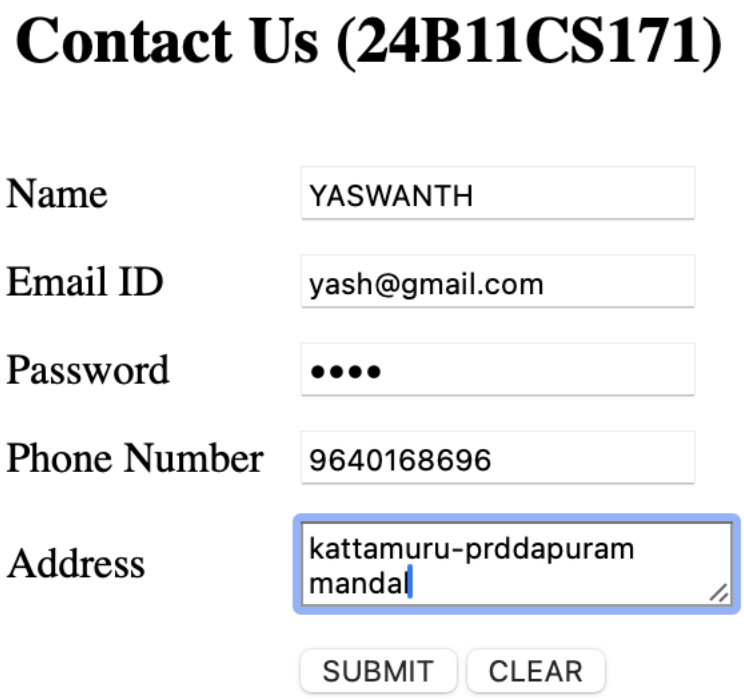
```
<form onsubmit="return validation()" action="registration_app.php" method="POST">
    <table align="center" border="0" cellpadding="6">
        <tr>
            <td colspan="2" align="center"><h2>Contact Us (24B11CS171)</h2></td>
        </tr>
        <tr>
            <td>Name</td>
            <td><input type="text" name="name" id="name_id" /></td>
        </tr>
        <tr>
            <td>Email ID</td>
            <td><input type="text" name="email" id="email_id" /></td>
        </tr>
        <tr>
            <td>Password</td>
            <td><input type="password" name="password" id="password_id" /></td>
        </tr>
        <tr>
            <td>Phone Number</td>
            <td><input type="text" name="phone" id="phone" /></td>
        </tr>
        <tr>
            <td>Address</td>
            <td><textarea name="address" id="address"></textarea></td>
        </tr>
        <tr>
            <td></td>
            <td>
                <input type="submit" value="SUBMIT" />
                <input type="reset" value="CLEAR" />
```

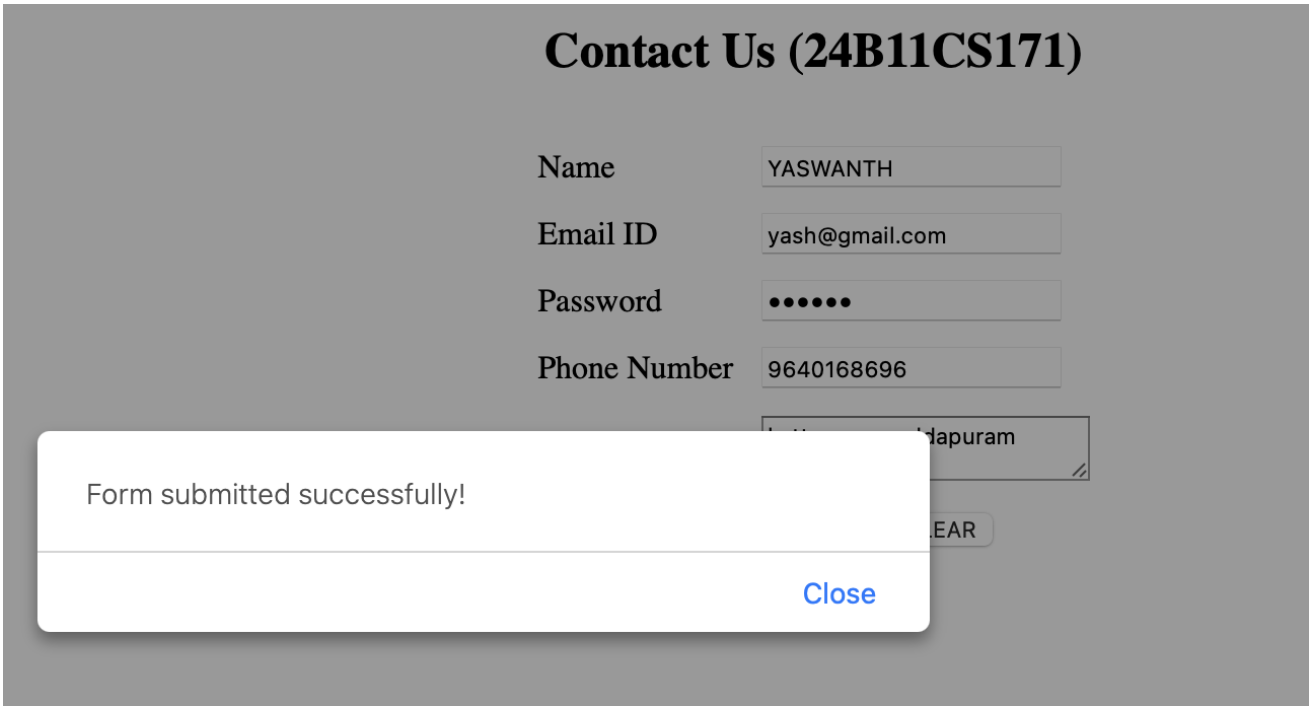
```
            </td>
          </tr>
        </table>
    </form>
</body>
</html>
```

Output:




Contact Us (24B11CS171)
Name
Email ID
Password
Phone Number
Please fill in all fields
Close


Contact Us (24B11CS171)
Name
YASWANTH
Email ID
yash@gmail.com
Password
Phone Number
9640168696
Address
kattamuru-prddapuram
mandal
SUBMIT
CLEAR


Contact Us (24B11CS171)
Name
YASWANTH
Email ID
yash@gmail.com
Password
Phone Number
9640168696
Form submitted successfully!
Close

Result:

Expno:8a

Aim: 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

Description:

This program demonstrates how to use JavaScript to compare multiple user-entered values and determine the largest among them. When the webpage loads, the program:

1. **Prompts the user** to enter three integers using prompt().
2. **Converts the input** values into integers using parseInt().
3. **Compares the numbers**:
    - If all three numbers are the same, it sets the message to **"Equal Numbers"**.
    - Otherwise, it uses the built-in method Math.max() to find the **largest number** and prepares a message showing that number followed by the words **"Larger Number"**.
4. **Displays the result** in an information dialog box using alert().

This program highlights the use of:

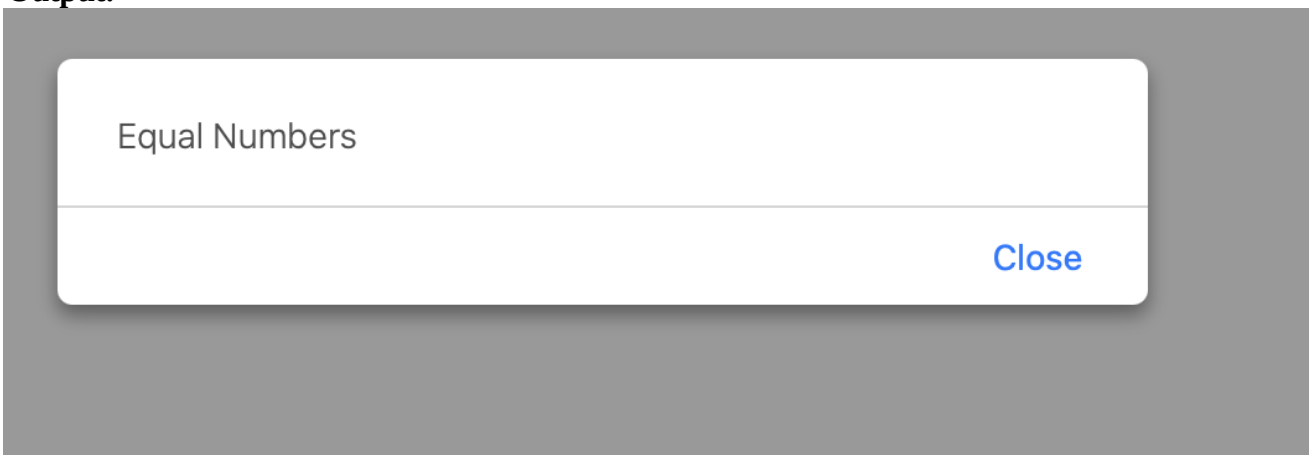
- User input handling with prompt().
- Number conversion using parseInt().
- Conditional statements (if...else) for decision-making.
- JavaScript's Math.max() function to find the maximum value.
- Output through an **alert box** to the user.

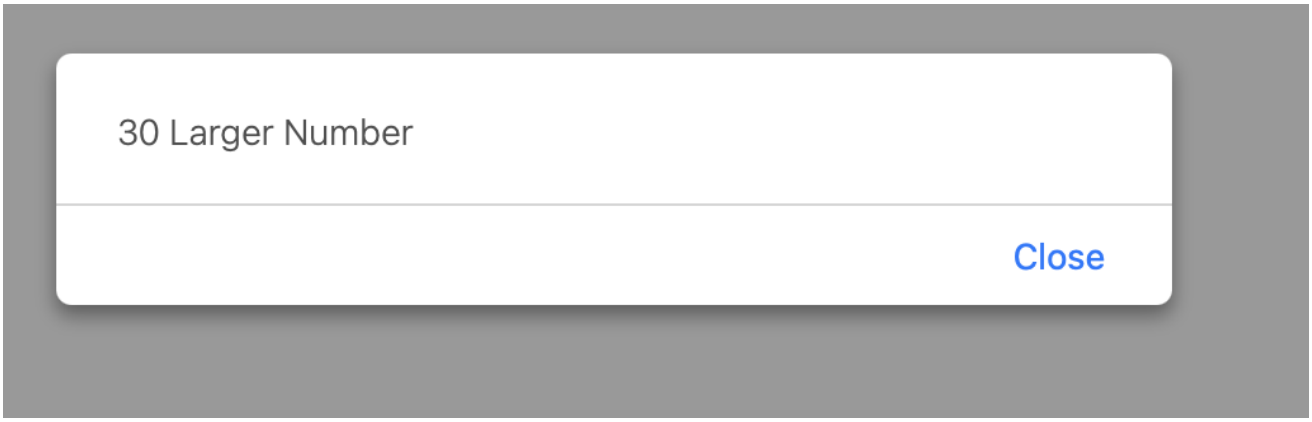
Program:

```
<html>
<head>
<title>Larger number finder</title>
<script>
function findLargerNumber(){
let num1=parseInt(prompt("enter first integer:"));
let num2=parseInt(prompt("enter second integer:"));
let num3=parseInt(prompt("enter third integer:"));
let message="";
```

```
if(num1===num2 && num2===num3){
message="Equal Numbers";
} else{
let largest=Math.max(num1,num2,num3);
message=largest + " Larger Number";
}
alert(message);
}
</script>
</head>
<body onload="findLargerNumber()">
</body>
</html>
```

Output:





Result:

Expno:8b

Aim: Write a program to display week days using switch case.

Description:

This program demonstrates the use of the **switch statement** in JavaScript to display the name of a weekday based on a number entered by the user.

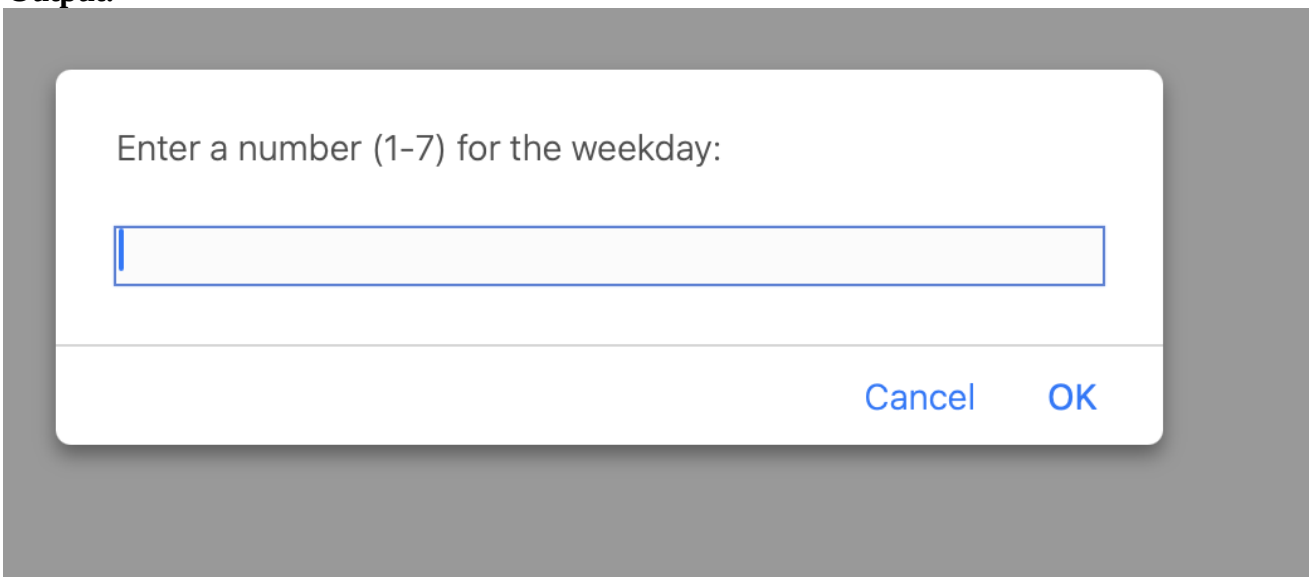
1. When the webpage loads, the function **showWeekDay()** is called.
2. The program uses a **prompt dialog box** to ask the user to enter a number between **1 and 7**.
    - 1 corresponds to **Sunday**
    - 2 corresponds to **Monday**
    - … and so on until 7, which corresponds to **Saturday**.
3. The **switch case** structure matches the entered number with the corresponding day name.
    - Each case assigns the correct day name to the variable dayName.
    - If the input is not between 1 and 7, the default case assigns the message **"Invalid input! please enter between 1-7"**.
4. Finally, the program displays the result inside the HTML element with the ID output.

Program:

```
<html>
<head>
<title>Week days with switch case</title>
<script>
function showWeekDay(){
   let dayNumber = parseInt(prompt("Enter a number (1-7) for the weekday:"));
   let dayName = "";
 switch(dayNumber){
      case 1: dayName = "Sunday"; break;
      case 2: dayName = "Monday"; break;
      case 3: dayName = "Tuesday"; break;
      case 4: dayName = "Wednesday"; break;
```

```
        case 5: dayName = "Thursday"; break;
        case 6: dayName = "Friday"; break;
        case 7: dayName = "Saturday"; break;
        default:
            dayName = "Invalid input! please enter between 1-7";
    }
  // Always show output
    document.getElementById("output").innerHTML = "<h2>" + dayName + "</h2>";
} </script>
</head>
<body onload="showWeekDay()">
    <p id="output"></p>
</body>
</html>
```

Output:



Thursday

Invalid input! please enter between 1-7

Result:

Expno:8C

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

Description:

For Loop

- **Syntax:**
- for(initialization; condition; increment/decrement) {
- // code block
- }
- **How it works:**
    - The loop starts with an initial value (i = 1).
    - It checks the condition (i <= 10).
    - Executes the loop body if the condition is true.
    - Increments i and repeats until the condition becomes false.
- **Example:** Prints numbers 1 to 10 using a for loop.

While Loop

- **Syntax:**
- while(condition) {
- // code block
- }
- **How it works:**
    - The condition is checked **before** executing the loop.
    - If true, the loop body executes.
    - Variable j is incremented inside the loop.
    - Loop stops when condition becomes false.

Do-While Loop

- **Syntax:**
- do {
- // code block
- } while(condition);
- **How it works:**

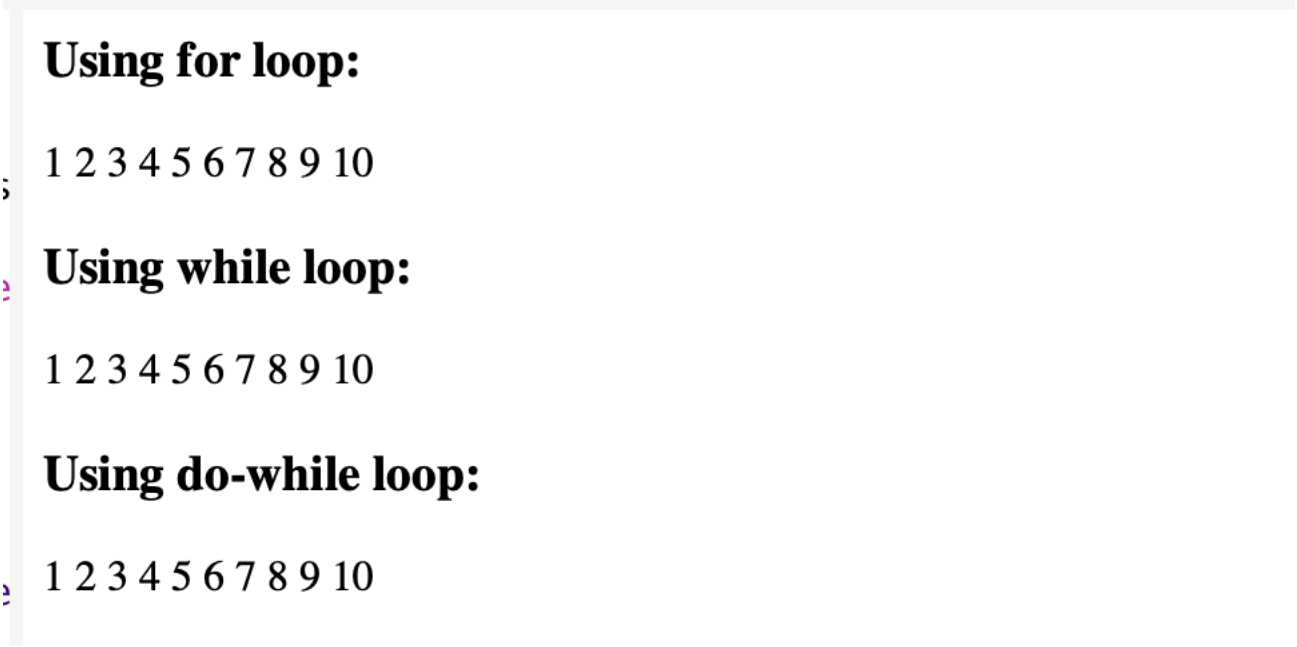
- The loop body executes **at least once**, even if the condition is false initially.
- After executing, the condition is checked.
- If true, it repeats the loop.

Program:

```
<!DOCTYPE html>
<html>
<head>
<title>Print 1 to 10 using Loops</title>
<script>
function printNumbers() {
    document.write("<h3>Using for loop:</h3>");
    for (let i = 1; i <= 10; i++) {
        document.write(i + " ");
    }

    document.write("<h3>Using while loop:</h3>");
    let j = 1;
    while (j <= 10) {
        document.write(j + " ");
        j++;
    }

    document.write("<h3>Using do-while loop:</h3>");
    let k = 1;
    do {
        document.write(k + " ");
        k++;
    } while (k <= 10);
}
```

```
</script>
</head>
<body onload="printNumbers()">
</body>
</html>
```

Output:



Result:

Expno:8d

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

Description:

for-in loop

- **Purpose:** Used to iterate over the **keys** (properties) of an object.
- **Syntax:**
- for (let key in object) {
- // code
- }
- **How it works:**
    - The for-in loop gets one key at a time from the object.
    - You can access its value using object[key].

forEach() loop

- **Purpose:** Used to iterate through **arrays**, but can also be used on object keys (after converting them into an array using Object.keys()).
- **Syntax:**
- Object.keys(object).forEach(function(key) {
- // code
- });
- **How it works:**
    - Object.keys(person) returns an array of keys → ["name", "age", "city"].
    - The forEach() method then iterates through this array.

for-of loop

- **Purpose:** Used to iterate directly over **iterable objects** like arrays, strings, or Object.entries() (array of key-value pairs).
- **Syntax:**
- for (let [key, value] of Object.entries(object)) {
- // code
- }
- **How it works:**

- Object.entries(person) returns [["name", "Ram"], ["age", 25], ["city", "Hyderabad"]].
- The for-of loop gives both key and value pairs in each iteration.

Program:

```
<html>
<head>
<title>Iterate object in JavaScript</title>
<script>
function displayObjectData() {
    const person = {
        name: "Ram",
        age: 25,
        city: "Hyderabad"
    };

    let output = "";

    // Using for-in (loops over keys)
    output += "<h3>Using for-in loop:</h3>";
    for (let key in person) {
        output += key + ": " + person[key] + "<br>";
    }

    // Using forEach (on Object.keys)
    output += "<h3>Using forEach loop:</h3>";
    Object.keys(person).forEach(function(key) {
        output += key + ": " + person[key] + "<br>";
    });

    // Using for-of (on Object.entries)
```

```
    output += "<h3>Using for-of loop:</h3>";
    for (let [key, value] of Object.entries(person)) {
        output += key + ": " + value + "<br>";
    }

    document.getElementById("result").innerHTML = output;
}
</script>
</head>
<body onload="displayObjectData()">
<h2 style="text-align:center;">Display object data using loops</h2>
<div id="result" style="margin:20px; font-family:Arial; font-size:16px;"></div>
</body>
</html>
```

Output:

Display object data using loops

Using for-in loop:

name: yash
age: 22
city: andhra

Using forEach loop:

name: yash
age: 22
city: andhra

Using for-of loop:

name: yash
age: 22
city: andhra

Result: