WEEK 4:

Selector forms

4.a. Write a program to apply different types of selector forms

4.a)i. Simple selector **(element, id, class, group, universal).**

**AIM:** Simple selector (element, id, class, group, universal).

**DESCRIPTION:**

**1. Element (Type) Selector**

Selects all elements of a specific type or tag in the HTML document.

**2. ID Selector**

Selects a single element that has a specific unique id attribute.

**3. Class Selector**

Selects all elements that share a specific class attribute.

**4. Group Selector**

Allows multiple selectors to be combined so that the same style applies to all of them.

**5. Universal Selector**

Selects all elements in the HTML document, regardless of type, class, or ID.

**CODE:**

<html>

<head>

<style>

P{

background-color:orange;

}

h1{

background-color:green;

</style>

</head>

<body>

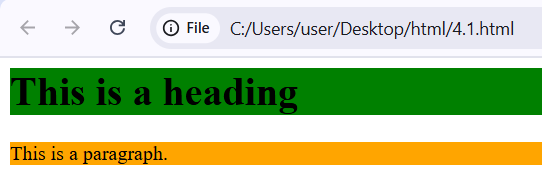
<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

**OUTPUT:**

****

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

**AIM:** Combinator selector (descendant, child, adjacent sibling, general sibling).

**DESCRIPTION:**

**1. Descendant Selector (A B)**

* Selects all elements **inside** a specified element, at **any level (nested deeply or not)**.
* Example meaning: div p → selects all <p> elements **inside** a <div>, no matter how deeply nested.

**2. Child Selector (A > B)**

* Selects only the elements that are **direct children** of a specified element.
* Example meaning: div > p → selects <p> elements that are **directly inside** a <div>, but not nested deeper.

**3. Adjacent Sibling Selector (A + B)**

* Selects an element that is **immediately after** a specified element (same parent).
* Example meaning: h1 + p → selects the **first <p>** right after an <h1>.

**4. General Sibling Selector (A ~ B)**

* Selects all elements that are **siblings** of a specified element and come **after it**, not just the first one.
* Example meaning: h1 ~ p → selects **all <p> elements** that come after an <h1> (sharing the same parent).

**CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>CSS Combinator selectors</title>

<style>

/\* Descendent selector(div p) \*/

div p {

color: blue;

}

/\* Child selector(div > p) \*/

div > p {

background-color: lightyellow;

}

/\* Adjacent sibling selector (h2 + p) \*/

h2 + p {

color: red;

font-weight: bold;

}

/\* General sibling selector (h2 ~ p) \*/

h2 ~ p {

font-style: italic;

}

/\* Styling for visibility \*/

div, section {

border: 1px solid #ccc;

padding: 10px;

margin: 10px;

}

</style>

</head>

<body>

<h1>CSS Combinator selectors example</h1>

<div>

<p>Child of div (affected by both descendant & child selectors)

</p>

<section>

<p>Descendant of div (only affected by descendant selector)</p>

</selection>

</div>

<h2>Heading (h2)</h2>

<p>Adjacent sibling (immediately after h2, styled red & bold)</p>

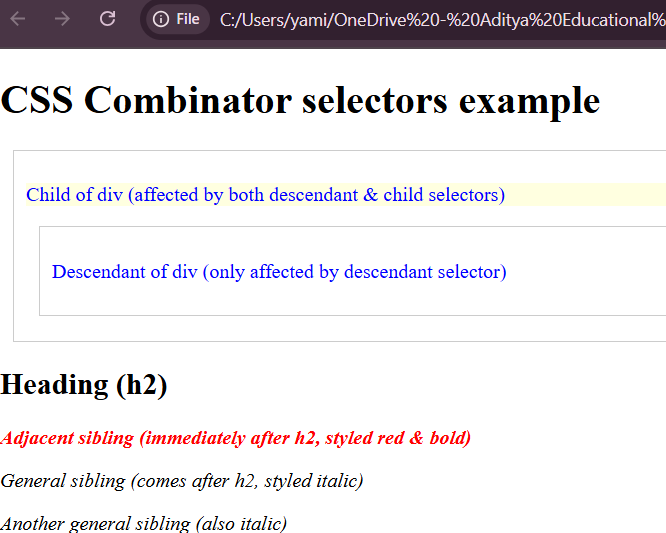
<p> General sibling (comes after h2, styled italic)</p>

<p>Another general sibling (also italic)</p>

</body>

</html>

**OUTPUT:**

****

**4.a)iii.** **Pseudo-class selector.**

**AIM:** Pseudo-class selector.

**DESCRIPTION:**

**Pseudo-class Selector**

A **pseudo-class** defines the **special state of an element** that cannot be targeted using normal selectors.  
It is written using a **colon (:)** before the pseudo-class name.

**Common Types of Pseudo-classes**

1. **User Action States**
   * :hover → applies when the mouse pointer is over an element.
   * :active → applies when an element is being clicked.
   * :focus → applies when an element (like input field) is focused.
2. **Structural States**
   * :first-child → selects the first child of its parent.
   * :last-child → selects the last child of its parent.
   * :nth-child(n) → selects the nth child of its parent.
3. **Form-related States**
   * :checked → applies to checkboxes or radio buttons when checked.
   * :disabled → applies to form inputs that are disabled.
   * :required → applies to form inputs with the “required” attribute.
4. **Other Useful States**
   * :link → applies to hyperlinks that have not been visited.
   * :visited → applies to hyperlinks already visited.
   * :empty → selects elements with no children (including text).

**CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Pseudo-class selector Example</title>

<style>

/\* Links \*/

a:link {

color: blue;

}

a:visited {

color: purple;

}

a:hover {

color: red;

}

a:active {

color: green;

}

/\* Input focus \*/

input:focus {

border: 2px solid orange;

background-color: #f9f2d0;

}

/\* List Styling \*/

ul li:first-child {

color: crimson;

}

ul li:last-child {

color: teal;

}

ul li:nth-child(2) {

color: darkorange;

}

/\* checkbox \*/

input:checked + label {

color: darkgreen;

font-weight: bold;

}

</style>

</head>

<body>

<h2>Pseudo-class Selector Demo</h2>

<p><a href="#">This is a link</a></p>

<p><input type="text" placeholder="Click here to see :focus effect"></p>

<ul>

<li>First item</li>

<li>Second item</li>

<li>Third item</li>

</ul>

<p>

<input type="checkbox" id="check1">

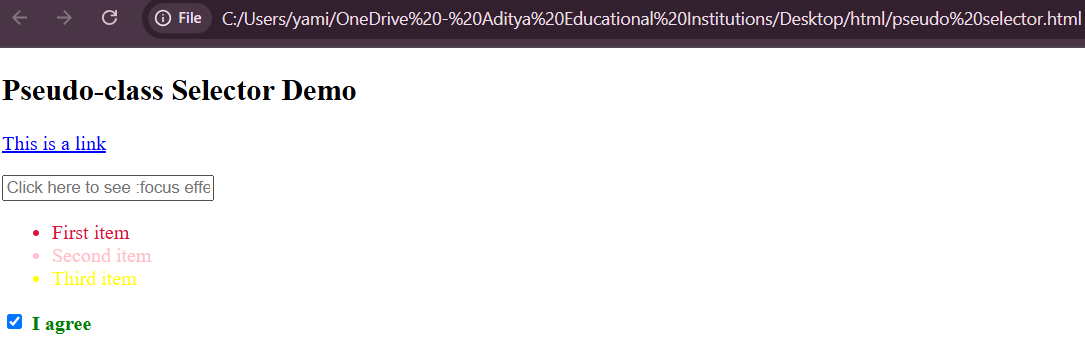
<label for="check1">I agree</label>

</p>

</body>

</html>

**OUTPUT:**

****

**4.a)iv.** **Pseudo-element selector.**

**AIM:** Pseudo-element selector.

**DESCRIPTION:**

**Pseudo-element Selector**

A **pseudo-element** is used to style **specific parts of an element’s content**.  
It is written with **two colons (::)** before the pseudo-element name (though older CSS versions allow a single colon :).

**Common Pseudo-elements**

1. **::before**
   * Inserts content **before** an element’s actual content.
   * Often used for adding icons, decorative text, or extra styling.
2. **::after**
   * Inserts content **after** an element’s actual content.
   * Commonly used with content: "" for design purposes (like clearfix or extra symbols).
3. **::first-letter**
   * Styles the **first letter** of an element.
   * Useful for drop caps in paragraphs.
4. **::first-line**
   * Styles the **first line** of an element’s text.
   * Works well for headings or paragraphs.
5. **::selection**
   * Styles the portion of text that a user has **highlighted/selected**.
6. **::marker**
   * Styles the **list item marker** (bullets, numbers) in lists.
7. **::placeholder**
   * Styles the placeholder text in input fields.

**CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Pseudo-Element Selectors 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: pink;

}

/\* ::after - insert content after an element \*/

h2::after {

content: " 👑";

color: pink;

}

/\* ::selection - style selected text \*/

::selection {

background-color:yellow;

color: black;

}

</style>

</head>

<body>

<h1>Pseudo-Element Selectors Demo</h1>

<h2>CSS Magic</h2>

<p>

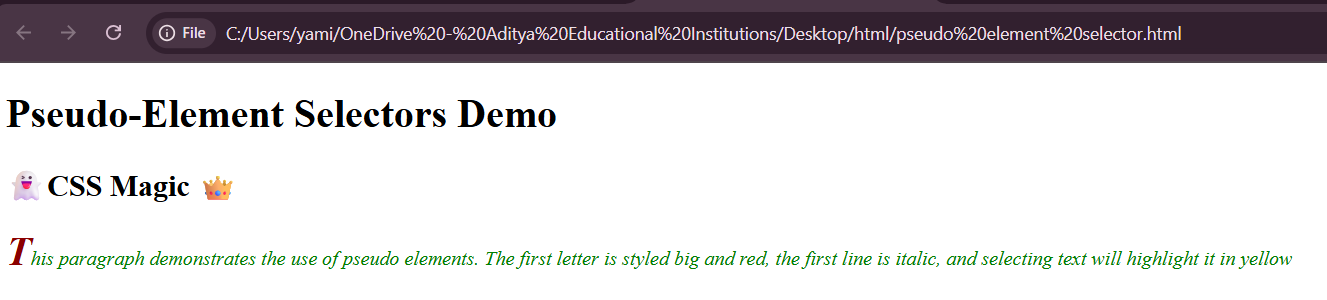
This paragraph demonstrates the use of pseudo elements. The first letter is styled big and red, the first line is italic, and selecting text will highlight it in yellow

</p>

</body>

</html>

**OUTPUT:**

****

**WEEK 5:**

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

**5a.** **Write a program to demonstrate the various ways you can reference a color in CSS.**

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

**DESCRIPTION:**

1. **Named Colors**
   * CSS has **147 predefined color names** like red, blue, green, gold, etc.
2. **Hexadecimal Colors (#RRGGBB)**
   * Uses **hex codes** where RR, GG, BB represent red, green, blue (values from 00 to FF).
   * Example: #ff0000 → red.
   * Can also be written in shorthand like #f00 → red.
3. **RGB Colors (rgb(r, g, b))**
   * Uses integer values **0–255** for red, green, blue.
   * Example: rgb(255, 0, 0) → red.
4. **RGBA Colors (rgba(r, g, b, a))**
   * Same as RGB but includes **alpha (opacity)** (0 = transparent, 1 = opaque).
   * Example: rgba(255, 0, 0, 0.5) → semi-transparent red.
5. **HSL Colors (hsl(h, s%, l%))**
   * Uses **Hue (0–360)**, **Saturation (%)**, **Lightness (%)**.
   * Example: hsl(0, 100%, 50%) → red.
6. **HSLA Colors (hsla(h, s%, l%, a))**
   * Same as HSL but with **alpha (opacity)**.
   * Example: hsla(0, 100%, 50%, 0.5) → semi-transparent red.

**CODE:**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>CSS Color Reference Methods</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: tomato; /\* CSS Named Color \*/

}

/\* 2. Hexadecimal 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 transparency) \*/

.rgba-color {

background-color: rgba(255, 0, 0, 0.6); /\* Red with transparency \*/

}

/\* 5. HSL notation \*/

.hsl-color {

background-color: hsl(300, 76%, 72%); /\* Light purple \*/

}

/\* 6. HSLA notation (with transparency) \*/

.hsla-color {

background-color: hsla(39, 100%, 50%, 0.7); /\* Orange with transparency \*/

}

</style>

</head>

<body>

<h1>Different Ways to Reference Colors in CSS</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>

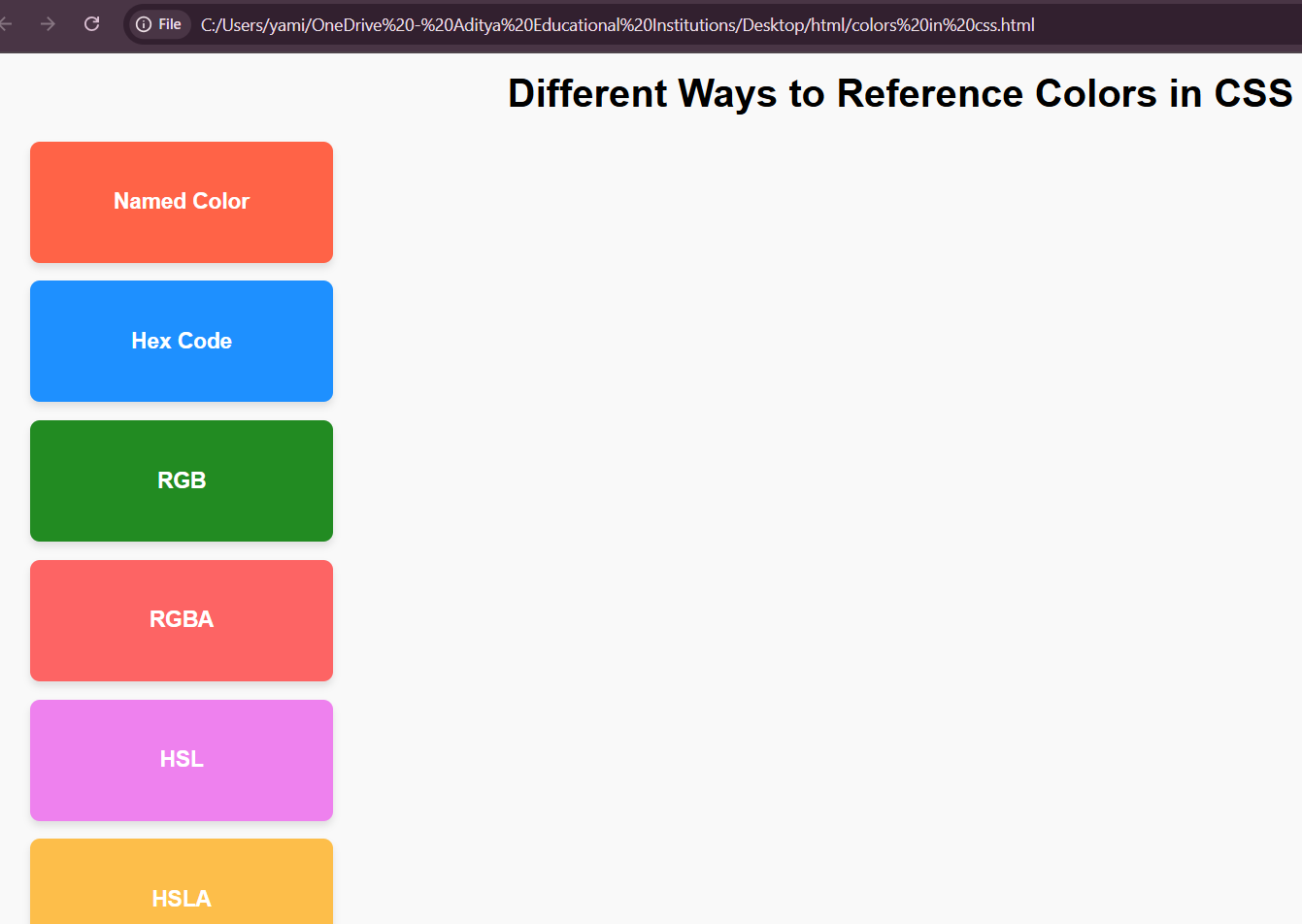
<div class="color-box hsl-color">HSL</div>

<div class="color-box hsla-color">HSLA</div>

</body>

</html>

**OUTPUT:**

****

**5b.** **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.**

**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:**

**Requirement Breakdown**

1. **Background image halfway down the page** → use background-position: center 50%;
   * center → horizontally centered.
   * 50% → vertically halfway down the page.
2. **Tilting it horizontally** → use transform: scaleX(-1); (flips the image horizontally).
   * Since background-image itself cannot be transformed directly, we usually apply it on a **pseudo-element** or container.
3. **Remain in place when scrolling** → use background-attachment: fixed;.

**CODE:**

<!doctype html>

<html>

<head>

<meta charset="utf-8" />

<title>Fixed tilted background (horizontal rotate)</title>

<style>

/\* Container content so you can scroll the page \*/

body {

min-height: 300vh; /\* make page scrollable to test 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 viewport \*/

transform: translate(-50%, -50%) rotateZ(-12deg) translateZ(0);

/\* translate to center, then rotate; translateZ(0) triggers GPU \*/

width: 60vw; /\* size the image element (adjust as needed) \*/

height: 40vh;

background-image:url("https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQC\_a55ixekvi-24aT8rf3\_7c7ggr1mX\_XpJuPKKQl67gxnUjVOzcLmf-7i6TJCEXJ5otaR&s");

background-size: cover;

background-position: center;

background-repeat: no-repeat;

pointer-events: none; /\* so it won't block clicks \*/

border-radius: 12px;

box-shadow: 0 10px 30px rgba(0,0,0,0.25);

z-index: 99

99; /\* keep it on top (adjust if necessary) \*/

capacity: 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</h1>

<p>Scroll to see the tilted background remain fixed halfway down the viewport.</p>

<p>Adjust <code>width</code>, <code>height</code>, and rotation angle in the CSS as needed.</p>

<!-- Add content to scroll -->

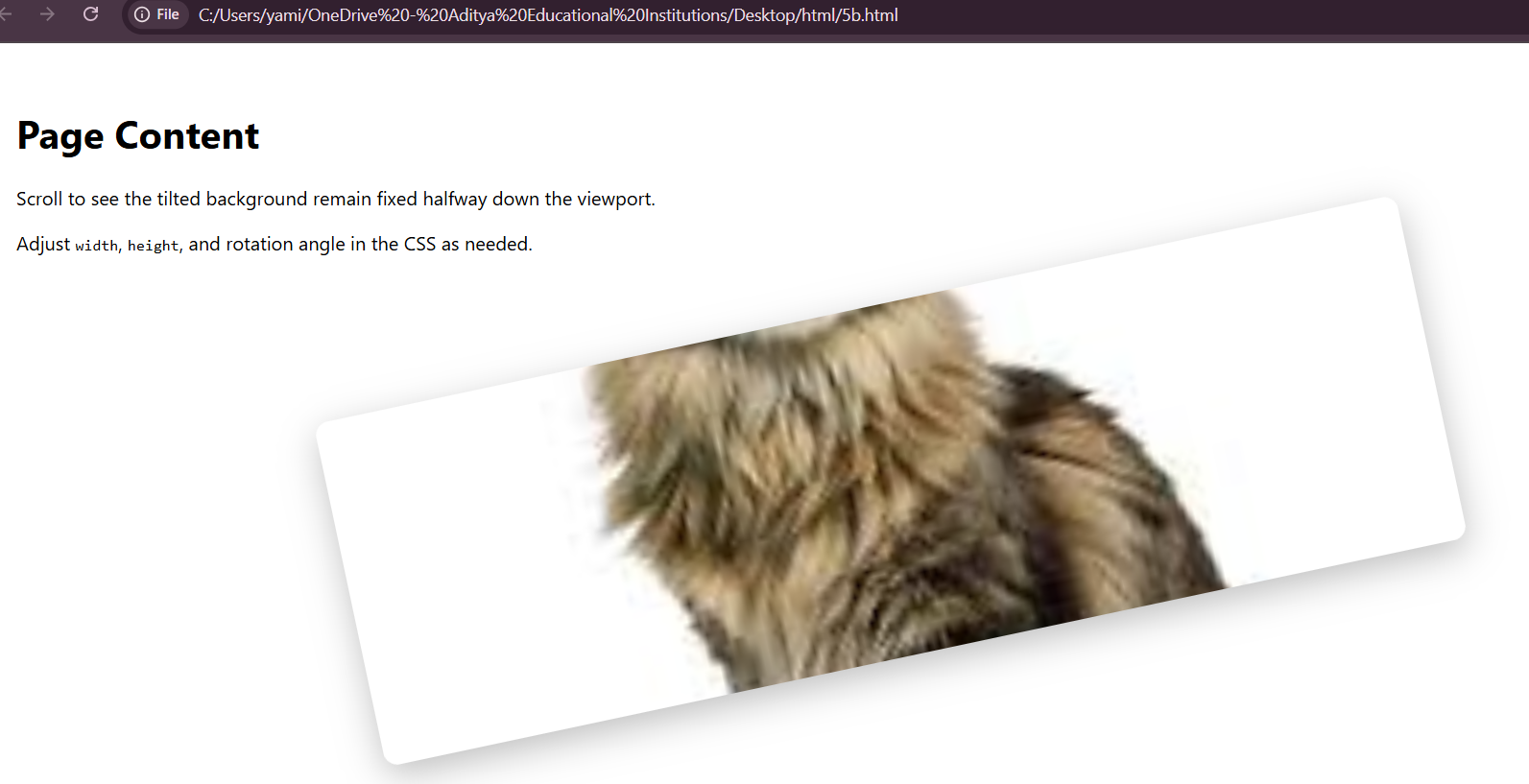
<p style="margin-top:140vh">This line is roughly below the fold to show scrolling.</p>

</main>

</body>

</html>

**OUTPUT:**

****

**5C.** **Write a program using the following terms related to CSS font and text: i. font-size ii. font-weight iv. text-decoration iii. font-style v. text-transformation vi. text-alignment.**

**AIM**: Write a program using the following terms related to CSS font and text: i. font-size ii. font-weight iv. text-decoration iii. font-style v. text-transformation vi. text-alignment.

**DESCRIPTION:**

1. **font-size**
   * Defines how large or small the text should appear.
   * Units: px, em, %, rem.
   * Example: font-size: 20px; makes the text 20 pixels tall.
2. **font-weight**
   * Specifies the thickness or boldness of the text.
   * Values: normal, bold, lighter, or numeric values from 100 to 900.
   * Example: font-weight: bold; makes text bold.
3. **font-style**
   * Defines the style of the text.
   * Values: normal (default), italic, oblique.
   * Example: font-style: italic; slants the text.
4. **text-decoration**
   * Adds decoration lines to the text.
   * Values: underline, overline, line-through, none.
   * Example: text-decoration: underline; draws a line below the text.
5. **text-transform**
   * Controls the capitalization of text.
   * Values: uppercase, lowercase, capitalize, none.
   * Example: text-transform: uppercase; converts text to capital letters.
6. **text-align**
   * Sets the horizontal alignment of text within its container.
   * Values: left, right, center, justify.
   * Example: text-align: center; centers the text in the page.

**CODE:**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>CSS font & 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 all letters 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 demonstration</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>

<p class="text-decoration">This text is underlined.</p>

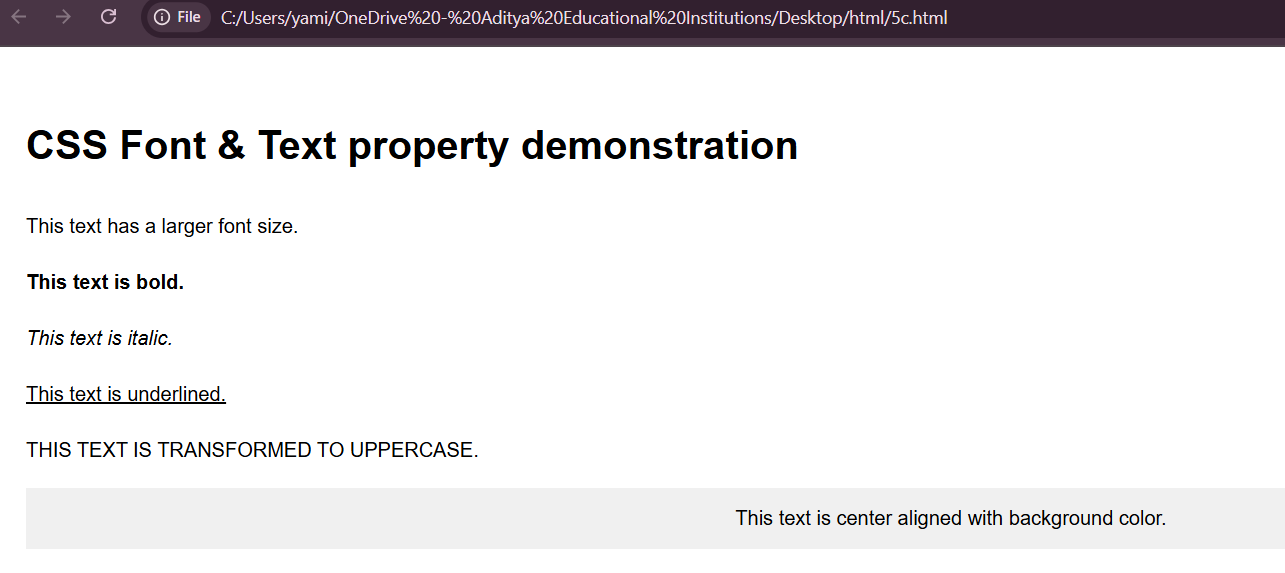
<p class="text-transform">This text is transformed to uppercase.</p>

<p class="text-align">This text is center aligned with background color.</p>

</body>

</html>

**OUTPUT:**

****

**5d.** **Write a program, to explain the importance of CSS Box model using i. Content ii. Border iii. Margin iv. Padding.**

**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** describes how every HTML element is treated as a rectangular box. It is made up of the following parts:

1. **Content**
   * The actual text, image, or element inside the box.
   * Controlled by width and height.
   * Example: "This is the content" in the program.
2. **Padding**
   * The transparent space **between content and border**.
   * Increases space inside the box without affecting the border.
   * Example: padding: 20px;
3. **Border**
   * A line around the content + padding.
   * Defines the visible boundary of the element.
   * Example: border: 5px solid blue;
4. **Margin**
   * The space **outside the border**.
   * Creates distance between this box and other elements.
   * Example: margin: 30px;

**CODE:**

<head>

<meta charset="UTF-8">

<title>CSS Box Model Example</title>

<style>

.box{

width:250px;

height:120px;

background-color:lightyellow; /\*i. content \*/

padding:20px; /\*iv. padding \*/

border:5px solid darkblue; /\*ii. border \*/

margin:30px; /\*iii. margin \*/

font-size:16px;

text-align:center;

}

body{

background-color: #f5f5f5;

font-family: Arial, sans-serif;

}

</style>

</head>

<body>

<h2 style="text-align:center;">CSS Box Model Demonstration</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>

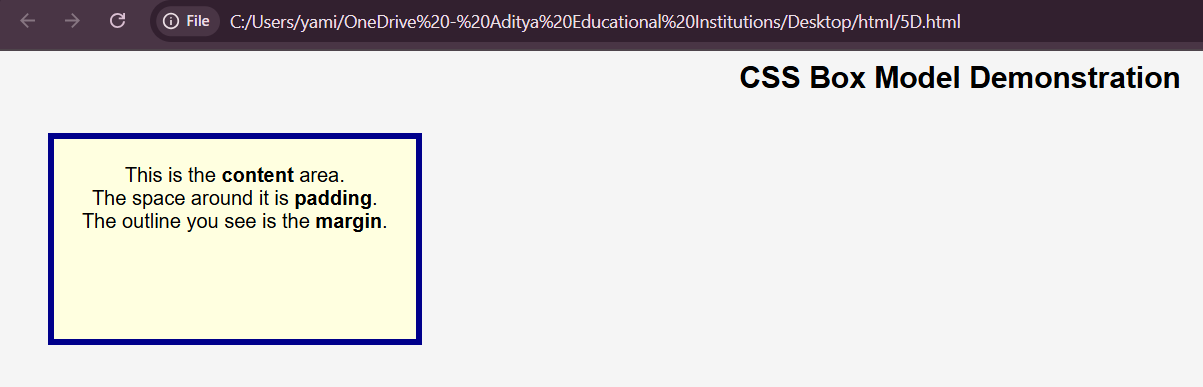
The empty space outside is the <b>margin</b>.

</div>

</body>

</html>

**OUTPUT:**

****

**WEEK :6**

**Applying JavaScript – internal and external, I/O, Type Conversion.**

**6a.** **Write a program to embed internal and external JavaScript in a web page.**

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

**DESCRIPTION:**

 **Internal JavaScript**

* Written inside the same HTML file using the <script>...</script> tag.
* Usually placed inside the <head> or <body> section.
* Example:
* <script>
* alert("Hello from Internal JS!");
* </script>

 **External JavaScript**

* Written in a separate .js file.
* Linked to the HTML file using the src attribute of the <script> tag.
* Example:

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

**CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>JavaScript Embedding Example</title>

<!--External JavaScript-->

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

</head>

<body>

<h1>Welcome to JavaScript Demo</h1>

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

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

<!--Internal JavaScript-->

<script>

function internalFunction

(){

alert("Hello from internal JavaScript");

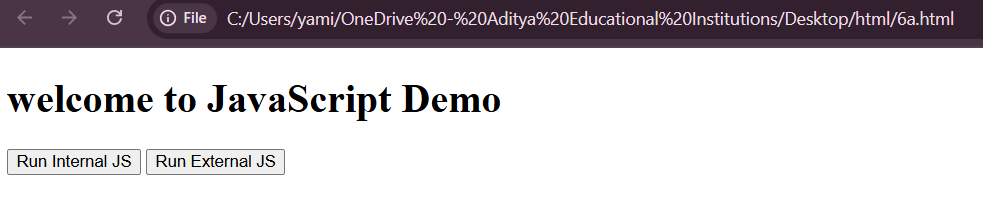
}

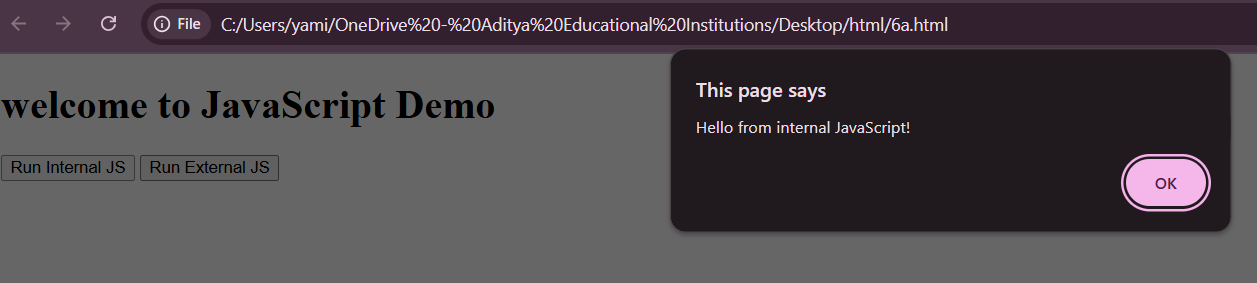
</script>

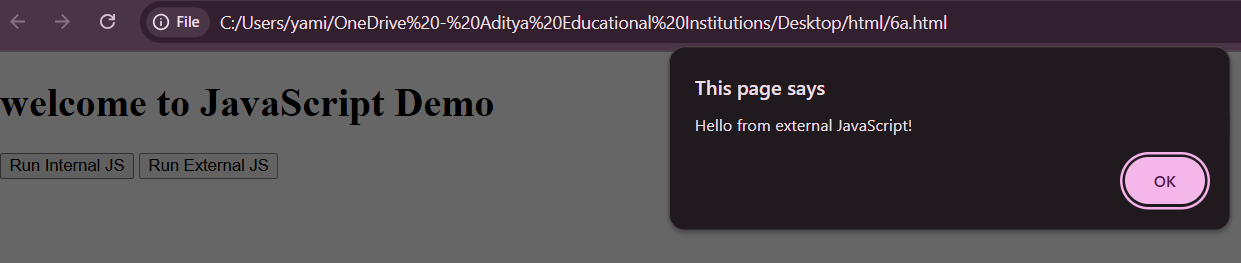
</body>

</html>

**OUTPUT:**

****

****

****

**6b.** **Write a program to explain the different ways for displaying output.**

**AIM:** Write a program to explain the different ways for displaying output.

**DESCRIPTION:**

**alert()**

* Shows output in a **popup dialog box**.
* Useful for giving messages or warnings to the user.
* Example: alert("Hello World!");

**console.log()**

* Prints output to the **browser console** (press F12 → Console).
* Mainly used for **debugging** and testing code.
* Example: console.log("Debug message");

**document.write()**

* Writes content directly into the web page.
* Example: document.write("This is output on the page");
* If used after the page is loaded, it can overwrite the entire page.

**innerHTML**

* Inserts output inside a specific HTML element.
* Example:
* document.getElementById("demo").innerHTML = "Hello!";
* This is the **most common method** for displaying output dynamically.

**window.print()** (optional)

* Opens the print dialog to print the current web page.
* Example: window.print();

**CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>JavaScript Output Methods</title>

<style>

body { font-family: Arial, sans-serif; padding: 20px; }

#outputArea { margin-top: 20px; padding: 10px; border: 1px solid #ccc; }

</style>

</head>

<body>

<h1>JavaScript Output Methods</h1>

<button onclick="showAlert()">Show Alert</button>

<button onclick="logToConsole()">Log to Console</button>

<button onclick="writeToDocument()">Write to Document</button>

<button onclick="updateDOM()">Update DOM</button>

<div id="outputArea">Output will appear here...</div>

<script>

// 1. Alert box0

function showAlert

() {

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

// 2. Console log

function logToConsole() {

console.log("This message is logged to the console.");

}

// 3. Document write

function writeToDocument() {

document.write("This text is written directly to the document.");

}

// 4. DOM manipulation

function updateDOM() {

document.getElementById("outputArea").innerText = "This text was inserted using DOM manipulation.";

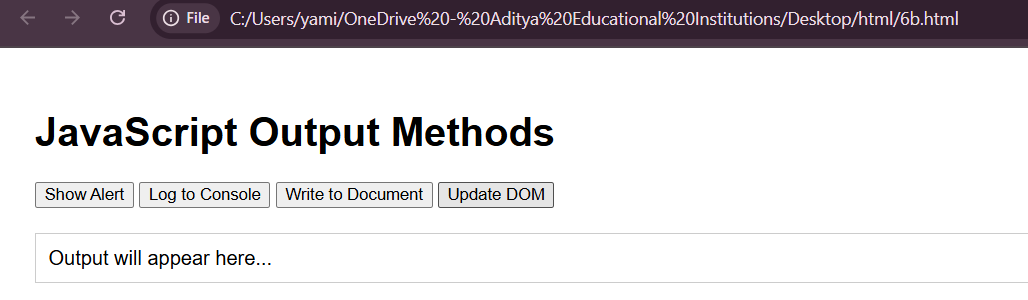
}

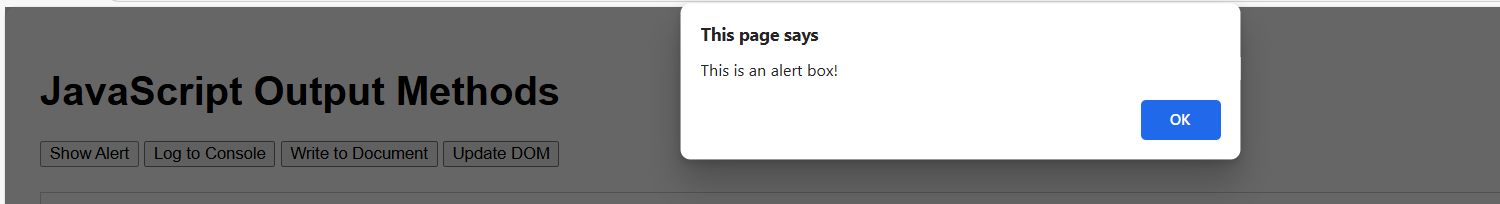
</script>

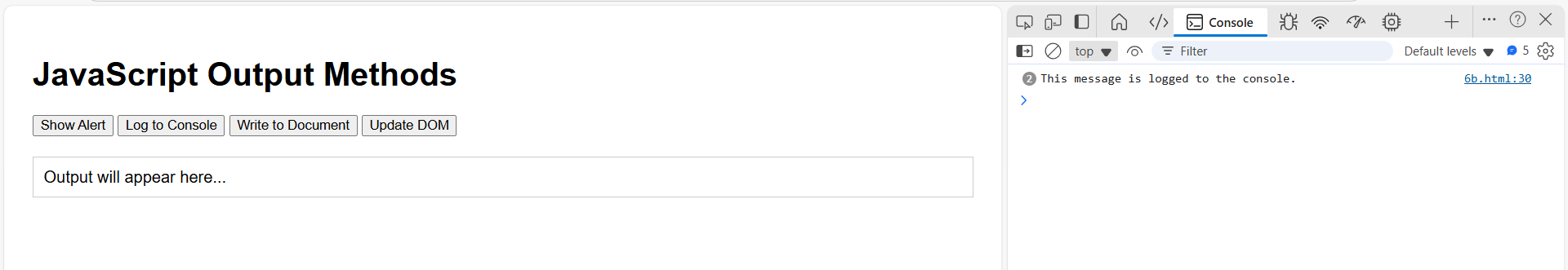
</body>

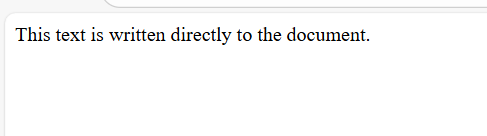
</html>

**OUTPUT:**

****

****

****

****

**6c.** **Write a program to explain the different ways for taking input.**

**AIM:**Write a program to explain the different ways for taking input.

**DESCRIPTION:**

**Different Ways of Taking Input in JavaScript**

1. **Prompt Box**
   * A small dialog box that asks the user to type something.
   * The entered value is then stored and can be used in the program.
   * Example use: Asking the user for their name or age.
2. **Confirm Box**
   * A dialog box with two buttons: **OK** and **Cancel**.
   * Used when you need a **Yes/No** or **True/False** type input.
   * Example use: Confirming if the user wants to exit a page or delete something.
3. **HTML Form Input Fields**
   * The most common way to take input in web pages.
   * Users type into text fields, checkboxes, radio buttons, or select options.
   * JavaScript can then collect these values and use them for processing.
4. **Event-based Input (Buttons, Key Press, Mouse Actions)**
   * Input can also come from user actions like clicking a button, pressing a key, or moving the mouse.
   * These events are captured and treated as input for interaction.

**CODE:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>JavaScript Input Methods</title>

<style>

body { font-family: Arial, sans-serif; padding: 20px; }

input,button{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-->

<button onclick="getinputFromField()">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>

<selectid="countrySelect">

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

<option value="USA">USA</option>

<option value="Japan">Japan</option>

</select>

<button onclick="getinputFromDropdown()">Submit Country</button>

<button onclick="updateDOM()">Update DOM</button>

<div id="output">your input will appear here...</div>

<script

// 1. Input from text field

function getInputFromField()

{

const name = document.getElementById("namelnput").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 for

function getInputFromForm (event)

{

event.preventDefault(); // Prevent page reload

const email = document.getElementById("emailInput").value;

document.getElementById("output").innerText = "Email: " + email;

}

//4.

function getinputFromDropdown()

{

const country = document.getElementById("");

document.getElementById("output").innerText = "Age: " + age;

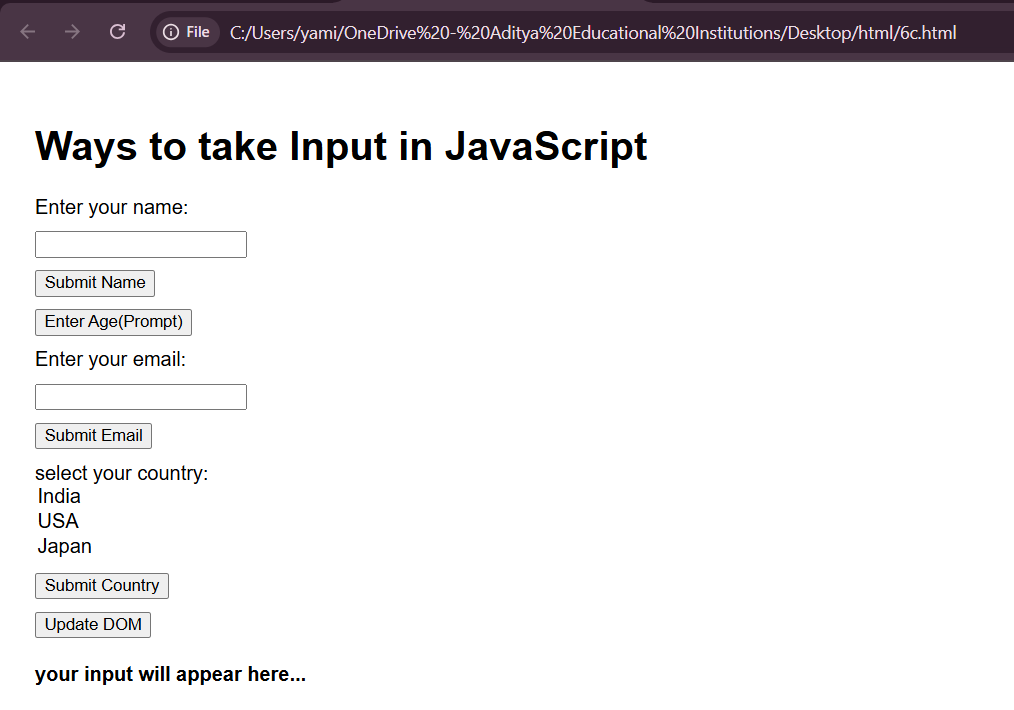
}

</script>

</body>

</html>

**OUTPUT:**

****

**6d.** **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.**

**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:**

This webpage demonstrates how to **take input using a prompt box** and display it in **table format**:

1. **Prompt for Name and Age**
   * The prompt() function asks the user to enter their **name** and **age**.
   * The entered age is converted to a number for comparison.
2. **Check Voting Eligibility**
   * A voter is eligible if age ≥ 18.
   * A simple conditional statement checks this and sets the eligibility message.
3. **Display in Table Format**
   * A dynamic HTML table is created with three columns: **Name, Age, Voting Eligibility**.
   * The table is inserted into the webpage using innerHTML.
4. **Interactive**
   * The user clicks a button to enter information and see results.

**CODE:**

<!DOCTYPE html>

<html>

<head>

<title>Voter Eligibility</title>

<style>

table {

border-collapse: collapse;

width: 50%;

margin-top: 20px;

}

th, td {

border: 1px solid black;

padding: 10px;

text-align: center;

}

th {

background-color: #f2f2f2;

}

h2 {

color: darkblue;

}

</style>

<script>

function getVoterInfo() {

// Ask for voter name

let name = prompt("Enter your name:");

// Ask for voter age

let age = prompt("Enter your age:");

age = Number(age); // Convert string to number

// Determine voting eligibility

let eligibility = age >= 18 ? "Eligible to Vote" : "Not Eligible to Vote";

// Create table HTML

let tableHTML = "<table>";

tableHTML += "<tr><th>Name</th><th>Age</th><th>Voting Eligibility</th></tr>";

tableHTML += `<tr><td>${name}</td><td>${age}</td><td>${eligibility}</td></tr>`;

tableHTML += "</table>";

// Display the table in the page

document.getElementById("voterTable").innerHTML = tableHTML;

}

</script>

</head>

<body>

<h2>Voter Eligibility Checker</h2>

<button onclick="getVoterInfo()">Enter Voter Information</button>

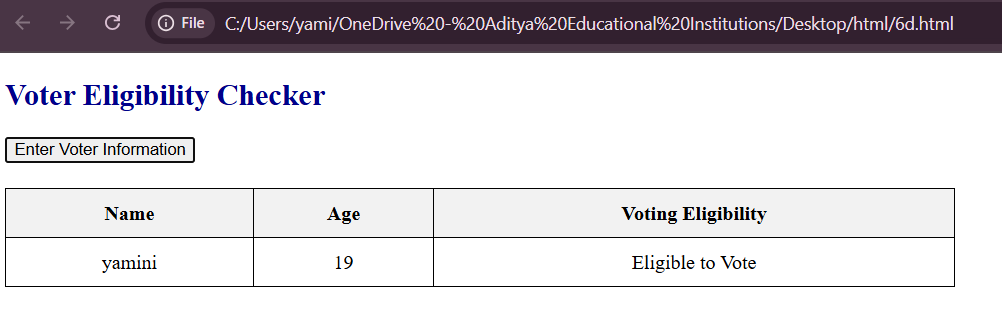
<!-- Placeholder for the table -->

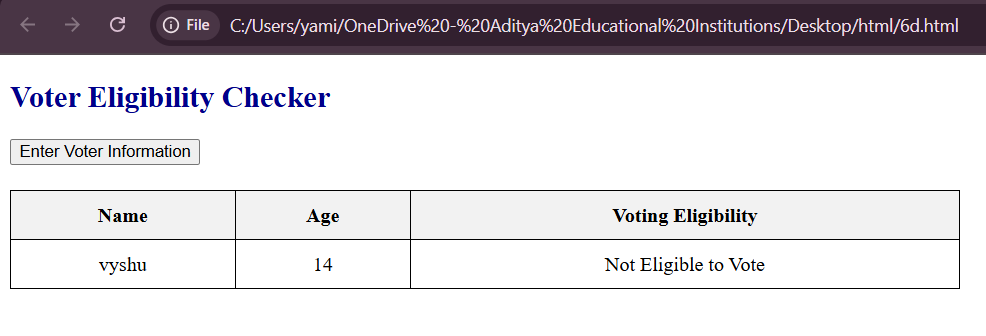
<div id="voterTable"></div>

</body>

</html>

**OUTPUT:**

****

****

**WEEK 7:**

**Javascript Pre-defined and User-defined Objects.**

**7a.** **Write a program using document object properties and methods.**

**AIM:** Write a program using document object properties and methods.

**DESCRIPTION:**

The **document object** in JavaScript represents the web page loaded in the browser. It provides **properties and methods** to interact with the page dynamically.

**1. Document Properties**

* **document.title** → Gets or sets the title of the document.
* **document.URL** → Returns the full URL of the document.
* **document.domain** → Returns the domain name of the document.
* **document.lastModified** → Returns the last modified date of the document.

**2. Document Methods**

* **document.getElementById()** → Access a specific HTML element by its ID.
* **document.write()** → Writes content directly into the HTML document. *(Should be used carefully after page load)*
* **document.body** → Access the <body> element to change its style or content.

**3. Importance**

* Enables **dynamic manipulation** of HTML content.
* Helps in **accessing page information** (URL, title, domain).
* Useful for **changing styles, content, and structure** of the web page programmatically.

**CODE:**

<html>

<head>

<title>Document Object Example</title>

</head>

<body>

<h1 id="heading">Hello World!</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 document methods

document.getElementById("heading").style.color = "blue";

//change heading color

document.getElementById("output").innerHTML =

"Title: " + title + "<br>"+

"URL: " + url + "<br>" +

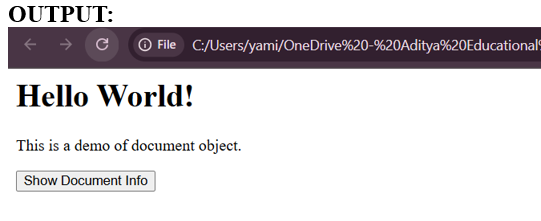
"Last Modified: " + lastModified;

}

</script>

</body>

</html>

****

**7b.** **Write a program using window object properties and methods.**

**AIM:** Write a program using window object properties and methods.

**DESCRIPTION:**

The **window object** represents the **browser window** and provides **properties and methods** to interact with it.

**1. Window Properties**

* **window.innerWidth** → Returns the width of the browser window viewport.
* **window.innerHeight** → Returns the height of the browser window viewport.
* **window.location.href** → Returns the URL of the current page.
* **window.history.length** → Returns the number of URLs in the browser session history.

**2. Window Methods**

* **window.alert()** → Displays a pop-up alert box.
* **window.confirm()** → Displays a pop-up with OK and Cancel buttons; returns true/false.
* **window.prompt()** → Displays a pop-up input box and returns user input.

**3. Importance**

* Enables **interaction with the browser window**.
* Useful for **displaying messages, taking input, and navigating pages**.
* Allows access to **screen size, URL, and session history** for dynamic web behavior.

**CODE:**

<!DOCTYPE html>

<html>

<head>

<title>Window Object Example</title>

</head>

<body>

<h1>Window Object Demo</h1>

<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();

}

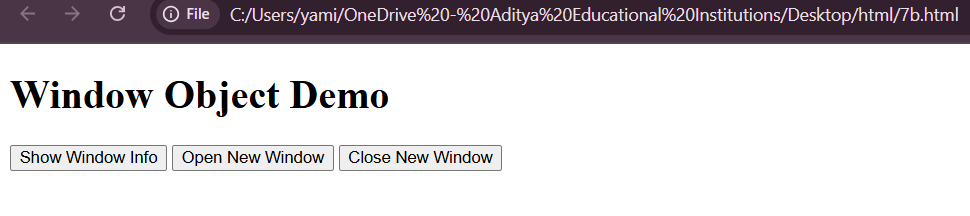
}

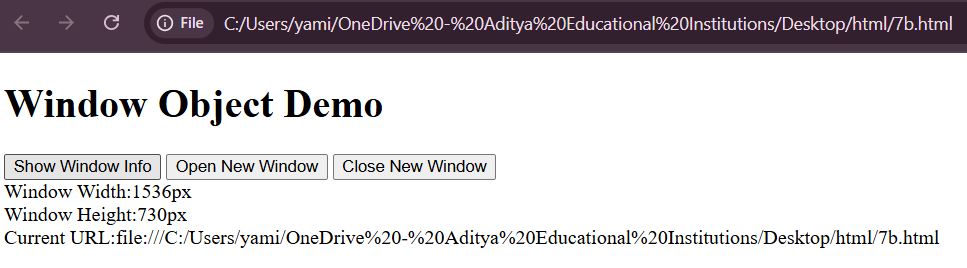
</script>

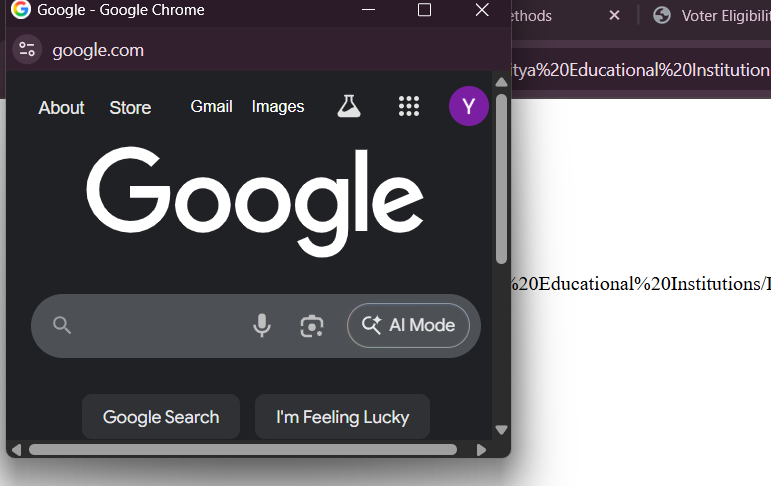
</body>

</html>

**OUTPUT:**

****

****

****

**7c.** **Write a program using array object properties and methods.**

**AIM:** Write a program using array object properties and methods.

**DESCRIPTION:**

The **Array object** in JavaScript allows us to store and manipulate **multiple values** in a single variable.

**1. Array Properties**

* **length** → Returns the number of elements in the array.
* Example:
* let fruits = ["Apple", "Banana"];
* console.log(fruits.length); // Output: 2

**2. Array Methods**

* **push(element)** → Adds element to the end of the array.
* **pop()** → Removes the last element from the array.
* **shift()** → Removes the first element.
* **unshift(element)** → Adds element to the beginning.
* **indexOf(element)** → Returns the index of the element. Returns -1 if not found.
* **reverse()** → Reverses the array elements.

**3. Importance**

* Arrays allow storing multiple values in an **ordered collection**.
* Array methods help in **adding, removing, searching, and rearranging elements** easily.
* They are **essential for handling lists of data** in JavaScript programs.

**CODE:**

DOCTYPE html>

<html>

<head>

<title>Array Object Properties and Methods</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.lenght;

//Array methods

fruits.push("Orange");

fruits.unshift("Grapes");

let last = fruits.pop();

let first = fruits.shift();

let sorted = fruits.sort();

let joined = fruits.join(" , ");

//Display results

document.getElementById("output").innerHTML=

"<b>Original Length:</b> " + last + "<br> +

"<b>After push & Unshift:</b> Apple,Banana,Mango,Orange,Grapes<br>"+"<b>Removed LAst:</b>"+last+"<br>"+"<b>"Removed First:</b>"+first+"<br>"+"<b>Joined as string:</b>"+joined;

</script>

</body>

</html>

**OUTPUT:**

****

**7d.** **Write a program using math object properties and methods.**

**AIM:** Write a program using math object properties and methods.

**DESCRIPTION:**

The **Math object** in JavaScript provides **mathematical constants and functions**.

**1. Math Properties**

* **Math.PI** → Returns the value of π (3.14159...).
* **Math.E** → Returns Euler’s number (2.718...).

**2. Math Methods**

* **Math.round(x)** → Rounds a number to the nearest integer.
* **Math.ceil(x)** → Rounds a number **up** to the next integer.
* **Math.floor(x)** → Rounds a number **down** to the previous integer.
* **Math.sqrt(x)** → Returns the square root of x.
* **Math.pow(x, y)** → Returns x raised to the power y.
* **Math.random()** → Returns a random number between 0 and 1.
* **Math.abs(x)** → Returns the absolute value of x.
* **Math.min(a, b, …)** → Returns the minimum value among the arguments.
* **Math.max(a, b, …)** → Returns the maximum value among the arguments.

**3. Importance**

* Provides **built-in mathematical calculations** without writing formulas manually.
* Useful in **games, random number generation, financial calculations, and scientific computations**.
* Helps in performing **rounding, power, root, min/max, and random operations** efficiently.

**CODE:**

<!DOCTYPE html>

<html>

<head>

<title>Math Object Properties and Methods</title>

</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 Euler's 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); //Round to nearest integer

let ceilResult=Math.ceil(4.2); //Round up

let floorResult=Math.floor(4.8); //Round down

let randomResult=Math.random(4.2); //Random number between 0 and 1

let maxResult=Math.max(3,9,2,15,7); //Maximum

let minResult=Math.min(3,9,2,15,7); //Minimun

//Display results

document.getElementById("output").innerHTML=

"<b>Properties:</b><br>"+

"PI:"+piValue+"<br>"+

"E:"+eValue+"<br>"+

"SQRT2:"+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>"+

"Max(3,9,2,15,7):"+maxResult+"<br>"+

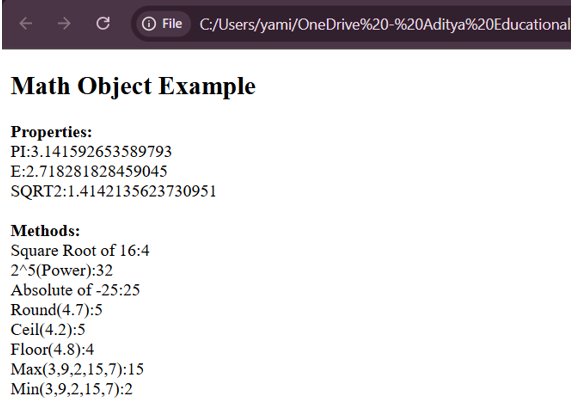
"Min(3,9,2,15,7):"+minResult;

</script>

</body>

</html>

**OUTPUT:**

****

**7e.** **Write a program using string object properties and methods.**

**AIM:** Write a program using string object properties and methods.

**DESCRIPTION:**

The **String object in JavaScript** is used to represent and manipulate sequences of characters. It provides several **properties** and **methods** to work with text.

**🔹 Description**

* **String Properties**
  + length: Returns the number of characters in the string.
* **String Methods (commonly used)**
  + charAt(index): Returns the character at the given index.
  + toUpperCase(): Converts the string to uppercase.
  + toLowerCase(): Converts the string to lowercase.
  + indexOf("sub"): Returns the first position of the given substring.
  + substring(start, end): Extracts part of the string between given indexes.
  + replace("old", "new"): Replaces a substring with another.
  + split(" "): Splits the string into an array based on a separator.
  + trim(): Removes whitespace from both ends of the string.
  + concat(): Joins two or more strings.

**CODE:**

<!DOCTYPE html>

<html>

<head>

<title>JavaScript string object properties and methods</title>

</head>

<body>

<h2>JavaScript String Object Example</h2>

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

<script>

//Creating a string

let text = "Hello JavaScript World!"

//Using String Properties

let length = text.length; //property

//Using String Methods

let upper=text.toUpperCase(); //convert to uppercase

let lower=text.toLowerCase(); //convert to lowercase

let trimmed=text.trim(); //remove spaces

let sliced=text.slice(6,16); //extract part of string

let replaced=text.replace("World","Programmers"); //replace word

let charAtPos=text.charAt(7); //get character at position

let index=text.indexOf("JavaScript"); //find index of word

let splitted=text.trim().split(" "); //split into array by spaces

//Display the results

document.getElementById("output").innerHTML=

"<b>Original String:</b>"+text+"<br>"+

"<>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:**

****

**7.f)** **Write a program using regex object properties and methods.**

**AIM:** Write a program using regex object properties and methods..

**DESCRIPTION:**

The RegExp (Regular Expression) object in JavaScript is used to define patterns for matching and searching text. It allows developers to test whether a string follows a particular format, extract specific parts of text, or replace matching patterns efficiently.

In this program, the RegExp object properties and methods are used to:

* Create a regular expression using either a literal (/pattern/flags) or the constructor (new RegExp("pattern", "flags")).
* Use methods such as:
  + test() → checks if a pattern exists in a given string and returns true or false.
  + exec() → searches a string for a match and returns the first match found (or null if none).
* Apply regular expressions to validate user inputs such as email addresses, phone numbers, or postal codes.
* Use string methods that support regex, like match(), replace(), and search(), for pattern-based operations.

This program demonstrates how the RegExp object can be used to perform pattern matching, validation, and text processing, making it a powerful tool or form validation and string manipulation in JavaScript.

**CODE:**

<html>

<head>

<title>Contact us Page</title>

<script type="text/javascript">

//Validation for fields

function validation(){

var nm=document.getElementById("name\_id").value;

var em=document.getElementById("email\_id").value;

var su=document.getElementById("subject\_id").value;

var me=document.getElementById("message\_id").value;

if(nm=="" && em=="" && su=="" && me=="")

{

alert("Please enter all fields");

return false;

}

else if(nm=="")

{

alert("Enter Name");

return false;

}

else if(nm.length<4)

{

alert("Name should not be below 6 character");

return false;

}

else if(em=="")

{

alert("Enter Email");

return false;

}

else if(!/^[A-z0-9]{6,}[@][A-z]{4,5}[.][A-z]{3}$/.test(em))

{

alert("Enter valid Email ID");

return false;

}

else if(su=="")

{

alert("Enter Subject");

return false;

}

else if(me=="")

{

alert("Enter message");

return false;

}

else if(ms.length<=20)

{

alert("message should below 20 characters ");

return false;

}

else{

return true;

}

}

</script>

</head>

<form action="login\_app.php" method="POST">

<table align="center">

<tr>

<td colspan="2"><h2>Contact us</h2></td>

</tr>

<tr>

<td>Email</td>

<td><input type="text" name="email" id="email"/></td>

</tr>

<tr>

<td>Password</td>

<td><input type="text" name="password" id="password\_id" /></td>

</tr>

<tr>

<td></td>

<td><input type="submit" value="SUBMIT" /><input type="reset" value="CLEAR" /></td>

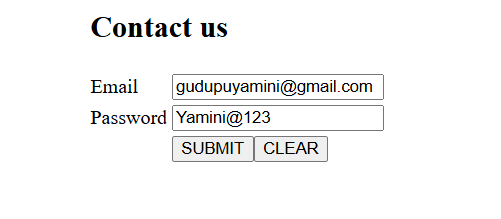
</tr>

</table>

</form>

</html>

**OUTPUT:**



**7.g) Write a program using date object properties and methods.**

**AIM:** Write a program using date object properties and methods.

**DESCRIPTION:**

The Date object in JavaScript is used to handle and manipulate dates and times. It allows developers to create new date instances, retrieve date and time information, and perform date-based calculations.

In this program, the Date object properties and methods are used to:

* Create a date object using new Date() to get the current date and time.
* Retrieve individual components of the date such as:
  + getDate() → returns the current day of the month.
  + getMonth() → returns the current month (0–11).
  + getFullYear() → returns the current year.
  + getDay() → returns the day of the week (0–6).
  + getHours(), getMinutes(), getSeconds() → return the current time details.
* Display the formatted date and time on the webpage.
* Optionally, modify date values using methods like setFullYear(), setMonth(), or setDate().

This program demonstrates how JavaScript’s Date object can be used to retrieve, format, and display date and time information dynamically, making webpages more interactive and time-aware.

**CODE:**

<!DOCTYPE html>

<html>

<head>

<title>Date Object Example</title>

</head>

<body>

<h2>JavaScript Date Object Example</h2>

<p id="dateInfo"></p>

<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>" +

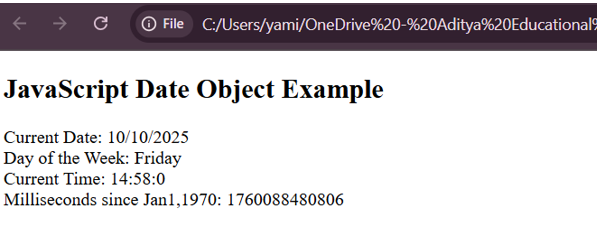
"Milliseconds since Jan1,1970: " + time;

</script>

</body>

</html>

**OUTPUT:**



**7h. Write a program to explain user-defined object by using properties, methods, accessors, constructors and display.**

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

**Description:**

A user-defined object in programming is a custom structure that groups data (called properties) and functions (called methods) together, representing a "real-world" entity. Here’s a detailed explanation of each key part—properties, methods, accessors, constructors, and a display method—with sample code and descriptions:

Description of Key Components

* Properties: Variables that belong to the object, representing its state or data (e.g., name, age in a Person object).
* Methods: Functions defined in the object that perform operations, often using or modifying the object’s properties (e.g., display()).
* Accessors: Special methods to "get" or "set" the values of properties, also known as getters and setters. They allow controlled access to private or protected data, and can provide computed values or validation.
* Constructor: A special method used to initialize a new object, setting up properties with initial values when the object is created.
* Display: A method that prints, returns, or otherwise presents the current state of the object—typically a summary of its properties, used for easy output.

**Code:**

<!DOCTYPE html>

<html>

<head>

<title>User-Defined Object Example</title>

</head>

<body>

<h2>User-Defined Object in JavaScript</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 return "Fail";

};

//Accessor(Setter)

this.setMarks=function(newMarks) {

this.marks=newMarks;

};

}

//Create an object using the constructor

let student1=new Student("Hasini",19,85);

student1.setMarks(92);

document.getElementById("output").innerHTML =

student1.displayDetails() + "<br>" +

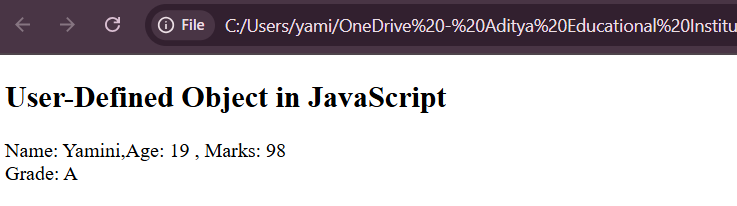
"Grade: "+student1.getGrade();

</script>

</body>

</html>

**Output:**

****

**Top of Form**

**Bottom of Form**