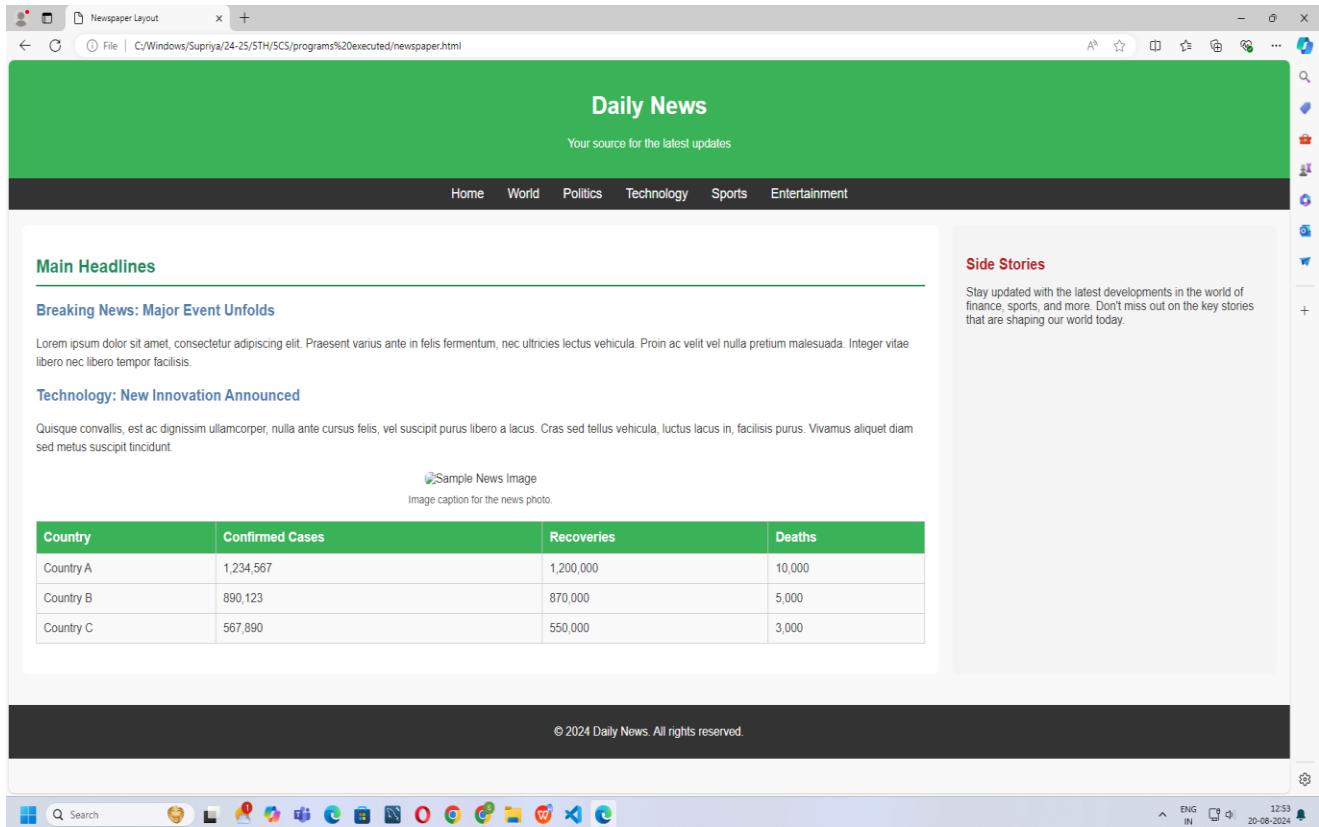


4. **Flexbox Layout:** The main element uses Flexbox to create a two-column layout with the article and sidebar.

Output:




```
.calculator input[type="number"] {  
  width: 90%;  
  padding: 10px;  
  margin-bottom: 10px;  
  border: none;  
  border-radius: 5px;  
  font-size: 18px;  
}
```

```
.calculator input[type="button"] {  
  width: 45%;  
  padding: 10px;  
  margin: 5px;  
  border: none;  
  border-radius: 5px;  
  font-size: 18px;  
  cursor: pointer;  
  background-color: #4CAF50;  
  color: white;  
}
```

```
.calculator input[type="button"]:hover {  
  background-color: #45a049;  
}
```

```
.calculator #result {  
  margin-top: 20px;  
  font-size: 24px;  
  color: yellow;  
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div class="calculator">
```

```
  <h2>Simple Calculator</h2>
```

```
  <input type="number" id="num1" placeholder="Enter first number">
```

```
  <input type="number" id="num2" placeholder="Enter second number (optional)">
```

```
<div>
```

```
  <input type="button" value="Sum" onclick="calculate('sum')">
```

```
  <input type="button" value="Product" onclick="calculate('product')">
```

```
  <input type="button" value="Difference" onclick="calculate('difference')">
```

```
<input type="button" value="Remainder" onclick="calculate('remainder')">
<input type="button" value="Quotient" onclick="calculate('quotient')">
<input type="button" value="Power" onclick="calculate('power')">
<input type="button" value="Square Root" onclick="calculate('sqrt')">
<input type="button" value="Square" onclick="calculate('square')">
</div>
```

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

```
<script>
function calculate(operation) {
    const num1 = parseFloat(document.getElementById('num1').value);
    const num2 = parseFloat(document.getElementById('num2').value);
    let result = "";

    switch(operation) {
        case 'sum':
            result = num1 + num2;
            break;
        case 'product':
            result = num1 * num2;
            break;
        case 'difference':
            result = num1 - num2;
            break;
        case 'remainder':
            result = num1 % num2;
            break;
        case 'quotient':
            result = num1 / num2;
            break;
        case 'power':
            result = Math.pow(num1, num2);
            break;
        case 'sqrt':
            result = Math.sqrt(num1);
            break;
        case 'square':
            result = num1 * num1;
            break;
        default:
            result = 'Invalid operation';
    }
}
```

```
}

document.getElementById('result').innerText = `Result: ${result}`;
}
</script>
</body>
</html>
```

Explanation of the Code

HTML Structure:

1. The calculator is enclosed within a <div> with the class calculator.
2. Two input fields (num1 and num2) are provided for entering numbers.
3. Buttons are provided for each operation (+, -, *, /, %, ^, √, x²).
4. A result field is provided (result) to display the calculation output.
5. A "Clear" button clears all input and output fields.

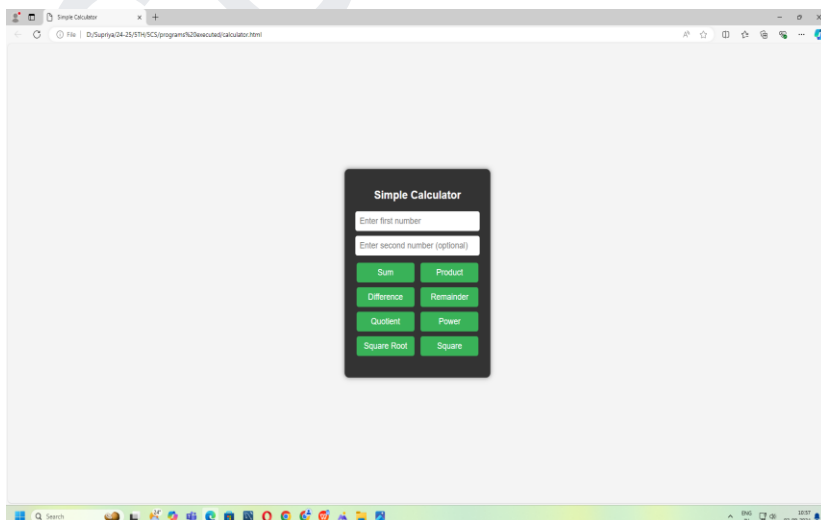
CSS Styling:

1. Basic styling is applied to center the calculator on the page and provide padding, margin, background color, and hover effects.
2. Flexbox is used to align elements and create a responsive layout.

JavaScript Functions:

1. The calculate() function takes an operation parameter to determine which calculation to perform.
2. It reads the values from the input fields, checks if they are valid numbers, and then performs the operation.
3. The result is displayed in the result input field.
4. The clearCalculator() function clears the input fields and result field.

Output:



Experiment-7

Develop JavaScript program (with HTML/CSS) for:

- a) Converting JSON text to JavaScript Object**
- b) Convert JSON results into a date**
- c) Converting From JSON To CSV and CSV to JSON**
- d) Create hash from string using crypto.createHash() method**

Source Code:

Step-by-Step Example

1. **Create an HTML file named json_to_object.html.**

Source Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Convert JSON to JavaScript Object</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
      background-color: #f0f0f0;
    }
    .container {
      max-width: 600px;
      margin: 0 auto;
      padding: 20px;
      background-color: white;
      border-radius: 5px;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    }
    textarea {
      width: 100%;
      height: 100px;
      padding: 10px;
      margin-bottom: 10px;
      font-size: 16px;
      border-radius: 5px;
```

```
border: 1px solid #ccc;
}
button {
padding: 10px 20px;
background-color: #007bff;
color: white;
border: none;
border-radius: 5px;
cursor: pointer;
font-size: 16px;
}
button:hover {
background-color: #0056b3;
}
pre {
background-color: #f7f7f7;
padding: 10px;
border-radius: 5px;
border: 1px solid #ddd;
font-size: 14px;
overflow-x: auto;
}
</style>
</head>
<body>
<div class="container">
<h2>Convert JSON to JavaScript Object</h2>
<textarea id="jsonInput" placeholder="Enter JSON text here...">{ "name": "John Doe",
"age": 30, "city": "New York" }</textarea>
<button onclick="convertJsonToObject()">Convert to Object</button>
<h3>Output:</h3>
<pre id="jsonOutput"></pre>
</div>
<script>
function convertJsonToObject() {
const jsonInput = document.getElementById('jsonInput').value;
try {
// Parse JSON string to JavaScript object
const jsonObject = JSON.parse(jsonInput);
// Convert the JavaScript object to a formatted JSON string for display
const formattedOutput = JSON.stringify(jsonObject, null, 2);

// Display the formatted JSON object
```

```
        document.getElementById('jsonOutput').textContent = formattedOutput;
    } catch (error) {
        // Display an error message if the JSON string is invalid
        document.getElementById('jsonOutput').textContent = 'Invalid JSON format. Please
check your input.';
    }
}
</script>
</body>
</html>
```

Explanation of the Code

HTML Structure:

1. A textarea element allows users to input JSON text. It comes pre-filled with a sample JSON object.
2. A button is provided to trigger the conversion process.
3. The pre element is used to display the formatted output or any error messages.

CSS Styling:

1. Basic CSS styling is added to enhance the user interface, making it more visually appealing and user-friendly.

JavaScript Functionality:

1. convertJsonToObject(): This function reads the JSON text from the textarea and attempts to parse it into a JavaScript object using JSON.parse().
2. If parsing is successful, it formats the JavaScript object using JSON.stringify() with indentation for better readability and displays the output.
3. If an error occurs during parsing (e.g., due to invalid JSON syntax), an error message is displayed.

Sample input:

```
{
  "name": "Alice",
  "age": 28,
  "city": "Wonderland",
  "isStudent": false
}
```

Output:

Convert JSON to JavaScript Object

```
{ "name": "John Doe", "age": 30, "city": "New York" }
```

Convert to Object

Output:

```
{  
  "name": "John Doe",  
  "age": 30,  
  "city": "New York"  
}
```

B)

Source Code:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Convert JSON to Date</title>  
  <style>  
    body {  
      font-family: Arial, sans-serif;  
      background-color: #f4f4f4;  
      padding: 20px;  
    }  
  
    #dateDisplay {  
      margin-top: 20px;  
      background-color: #333;  
      color: white;  
      padding: 10px;  
      border-radius: 5px;  
    }  
  </style>  
</head>  
<body>  
  <div id="dateDisplay">  
    <div id="date"></div>  
  </div>  
</body>  
</html>
```

```
</style>
</head>
<body>

  <h2>Convert JSON Results to Date</h2>
  <textarea id="jsonDataInput" rows="10" cols="50" placeholder="Enter JSON with date like
{"date":"2024-08-22T10:00:00Z"}"></textarea><br><br>
  <button onclick="convertToDate()">Convert to Date</button>

  <div id="dateDisplay"></div>

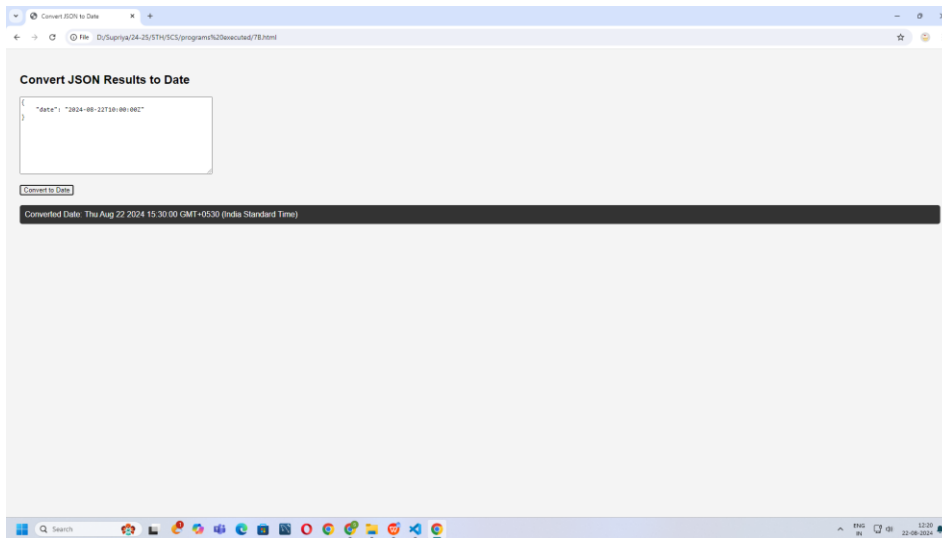
  <script>
    function convertToDate() {
      const jsonText = document.getElementById('jsonDataInput').value;
      try {
        const jsObject = JSON.parse(jsonText);
        const date = new Date(jsObject.date);
        document.getElementById('dateDisplay').innerHTML = `Converted Date:
${date.toString()}`;
      } catch (error) {
        document.getElementById('dateDisplay').innerHTML = "Invalid JSON format!";
      }
    }
  </script>

</body>
</html>
```

Sample input:

```
{
  "date": "2024-08-22T10:00:00Z"
}
```

Output:



C)

Source code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JSON to CSV and CSV to JSON</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f4;
      padding: 20px;
    }

    #csvJsonDisplay {
      margin-top: 20px;
      background-color: #333;
      color: white;
      padding: 10px;
      border-radius: 5px;
    }
  </style>
</head>
<body>

  <h2>Convert JSON to CSV and CSV to JSON</h2>

  <h3>JSON to CSV</h3>
  <textarea id="jsonToCsvInput" rows="10" cols="50" placeholder="Enter JSON array like
[{"name":"John","age":30},{"name":"Jane","age":25}]"></textarea><br><br>
```

```
<button onclick="convertJsonToCsv()">Convert to CSV</button>
```

```
<h3>CSV to JSON</h3>
```

```
<textarea id="csvToJsonInput" rows="10" cols="50" placeholder="Enter CSV text  
here..."></textarea><br><br>
```

```
<button onclick="convertCsvToJson()">Convert to JSON</button>
```

```
<div id="csvJsonDisplay"></div>
```

```
<script>
```

```
function convertJsonToCsv() {  
  const jsonText = document.getElementById('jsonToCsvInput').value;  
  try {  
    const jsObject = JSON.parse(jsonText);  
    const headers = Object.keys(jsObject[0]);  
    const csv = [  
      headers.join(','),  
      ...jsObject.map(row => headers.map(header => row[header]).join(','))  
    ].join('\n');  
    document.getElementById('csvJsonDisplay').innerHTML = `  } catch (error) {  
    document.getElementById('csvJsonDisplay').innerHTML = "Invalid JSON format!";  
  }  
}
```

```
function convertCsvToJson() {  
  const csvText = document.getElementById('csvToJsonInput').value;  
  try {  
    const [headers, ...rows] = csvText.split('\n');  
    const headerArray = headers.split(',');  
    const jsonArray = rows.map(row => {  
      const values = row.split(',');  
      return headerArray.reduce((obj, header, index) => {  
        obj[header] = values[index];  
        return obj;  
      }, {});  
    });  
    document.getElementById('csvJsonDisplay').innerHTML =  
    `  } catch (error) {  
    document.getElementById('csvJsonDisplay').innerHTML = "Invalid CSV format!";  
  }  
}
```

</script>

</body>

</html>

Sample input:

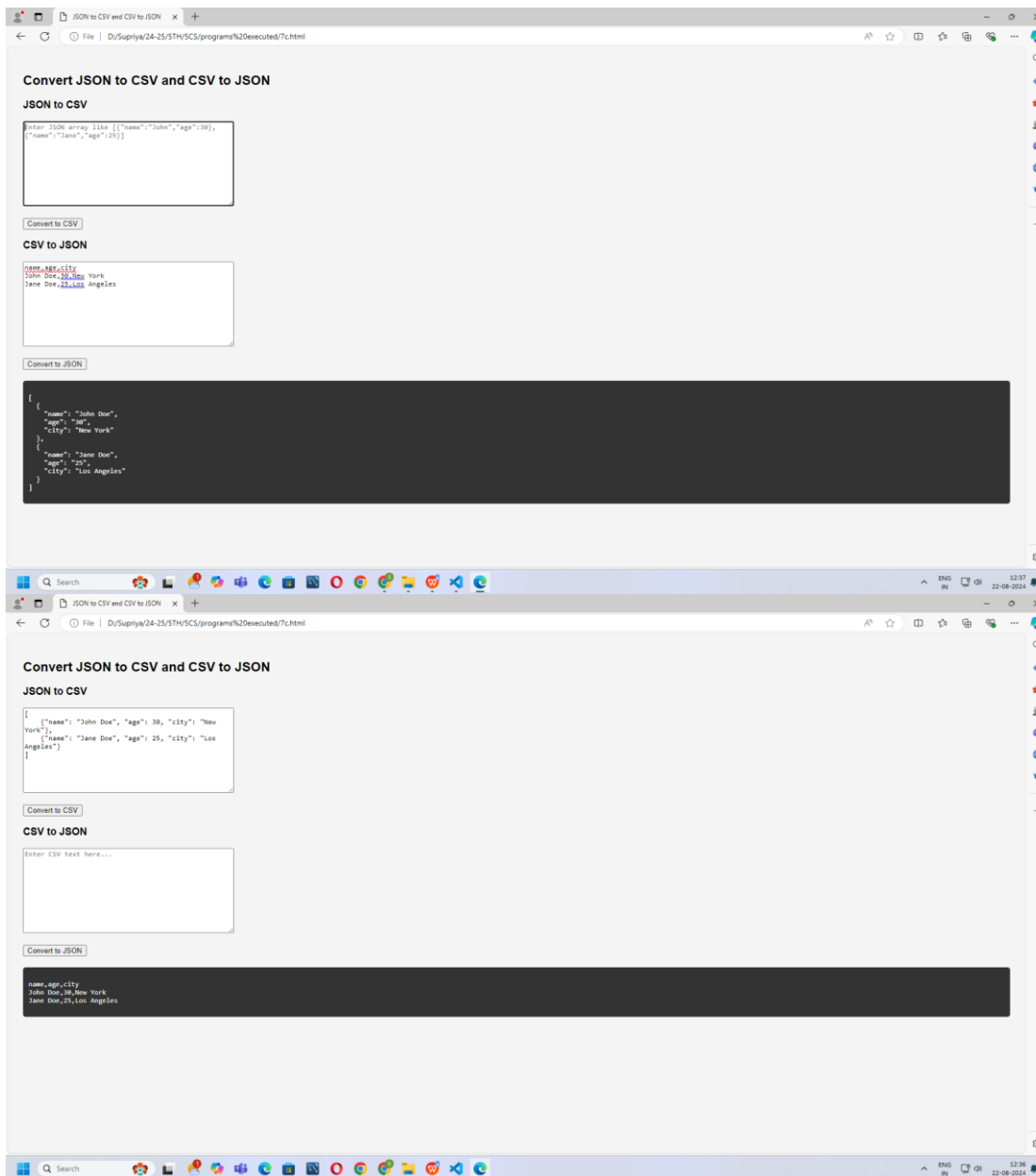
Sample Input for JSON to CSV

```
[  
  {"name": "John Doe", "age": 30, "city": "New York"},  
  {"name": "Jane Doe", "age": 25, "city": "Los Angeles"}  
]
```

Sample Input for CSV to JSON

```
name,age,city  
John Doe,30,New York  
Jane Doe,25,Los Angeles
```

Output:



D)

Step 1: Install Node.js

Ensure that Node.js is installed on your system. You can download and install it from the official Node.js website: <https://nodejs.org/>.

Step 2: Create a JavaScript File

Create a JavaScript file named `hashGenerator.js` with the following code:

Source Code:

```
// Import the crypto module  
const crypto = require('crypto');
```

```
// Function to create a hash from a string
function createHashFromString(inputString) {
  // Use the createHash method and specify the algorithm ('sha256', 'sha512', etc.)
  const hash = crypto.createHash('sha256');

  // Update the hash content with the input string
  hash.update(inputString);

  // Calculate the digest in hexadecimal format
  const hashOutput = hash.digest('hex');

  return hashOutput;
}

// Test the function
const inputString = 'Hello, World!';
const hashResult = createHashFromString(inputString);

console.log(`Input String: ${inputString}`);
console.log(`SHA-256 Hash: ${hashResult}`);
```

Sample input:

```
const input = 'Hello, World!';
```

Explanation of the Code

Importing the Crypto Module:

1. The crypto module is imported using `require('crypto')`.

Creating a Hash:

1. The `createHash` function from the crypto module is used to create a new hash object. The algorithm for hashing is specified ('sha256' in this example, but you can use others like 'md5', 'sha512', etc.).

Updating the Hash Content:

1. The `update` method is used to specify the input string for which the hash is to be generated.

Calculating the Digest:

1. The `digest` method computes the hash digest and returns it in hexadecimal format ('hex').

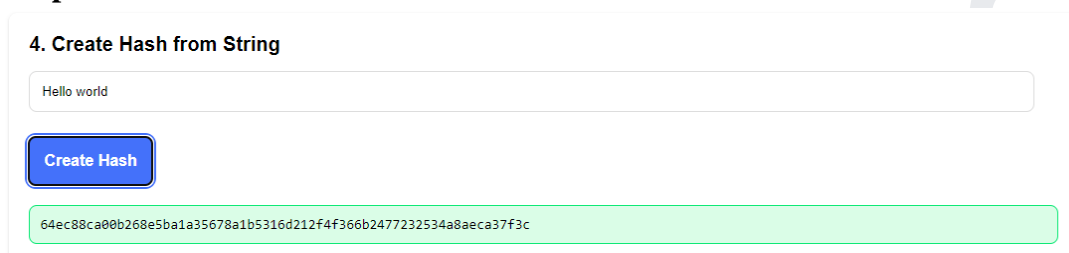
Testing the Function:

1. A sample input string 'Hello, World!' is passed to the function to generate its hash, and the result is printed to the console.

Step 3: Run the JavaScript File

To run the file, open a terminal or command prompt, navigate to the directory where hashGenerator.js is saved, and run the following command:

Output:



4. Create Hash from String

Hello world

Create Hash

64ec88ca00b268e5ba1a35678a1b5316d212f4f366b2477232534a8aeca37f3c

Experiment-8

- a. Develop a PHP program (with HTML/CSS) to keep track of the number of visitors visiting the web page and to display this count of visitors, with relevant headings.
- b. Develop a PHP program (with HTML/CSS) to sort the student records which are stored in the database using selection sort.

Source Code:

Experiment-9

Develop jQuery script (with HTML/CSS) for:

- Appends the content at the end of the existing paragraph and list.
- Change the state of the element with CSS style using animate() method
- Change the color of any div that is animated.

Source Code:

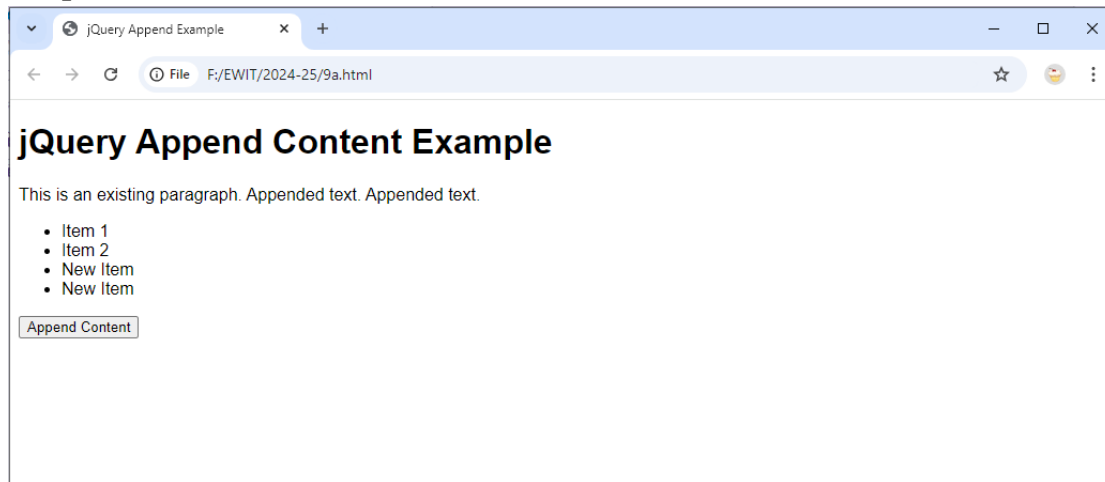
A)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>jQuery Append Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
    }
    #content {
      margin-top: 20px;
      padding: 10px;
      background-color: #e0f7fa;
      border-radius: 5px;
    }
  </style>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <h1>jQuery Append Content Example</h1>
  <p id="paragraph">This is an existing paragraph.</p>
  <ul id="list">
    <li>Item 1</li>
    <li>Item 2</li>
  </ul>
  <button id="appendContent">Append Content</button>

  <script>
    $(document).ready(function() {
      $('#appendContent').click(function() {
        $('#paragraph').append(' Appended text.');
```

```
        $('#list').append('<li>New Item</li>');
    });
});
</script>
</body>
</html>
```

Output:

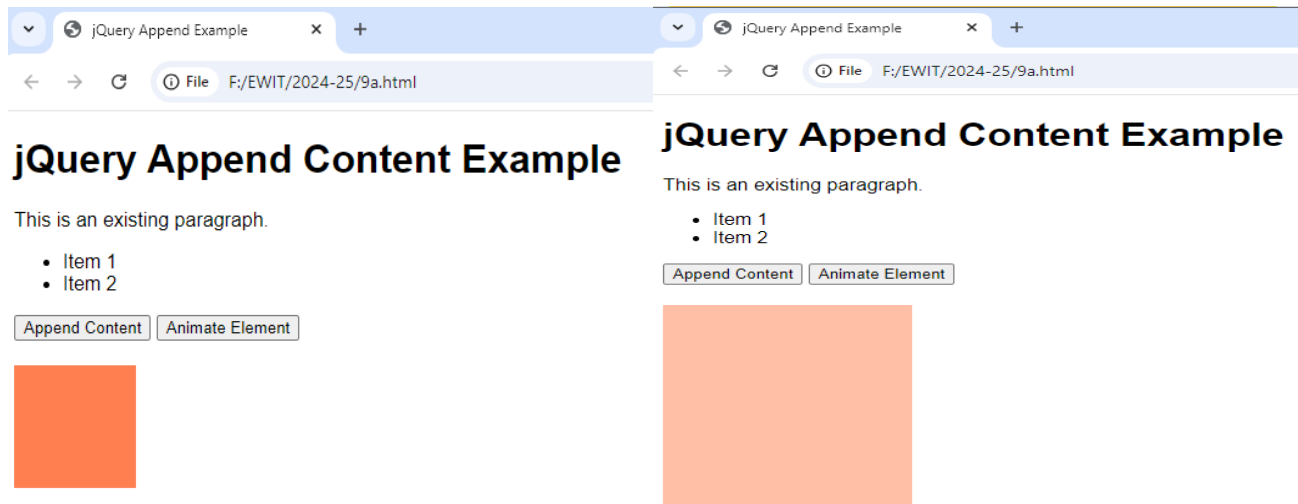


B)Additional Code for Animation:

```
<button id="animateElement">Animate Element</button>
<div id="animateDiv" style="width: 100px; height: 100px; background-color: coral; margin-top: 20px;"></div>

<script>
    $('#animateElement').click(function() {
        $('#animateDiv').animate({
            width: '200px',
            height: '200px',
            opacity: 0.5
        }, 1000);
    });
</script>
```

Output:

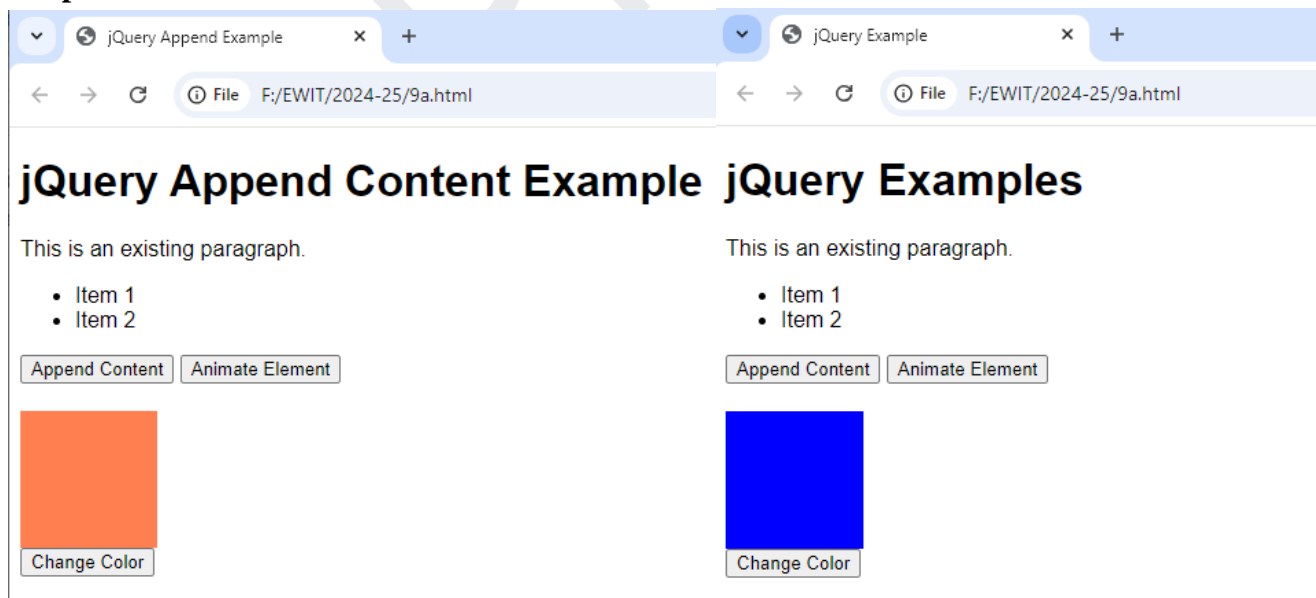


C)Additional Code for Changing Color

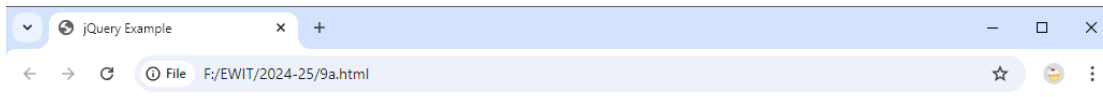
```
<button id="changeColor">Change Color</button>
```

```
<script>
    $('#changeColor').click(function() {
        $('#animateDiv').animate({ backgroundColor: 'blue' }, 1000);
    });
</script>
```

Output:



Final Output:



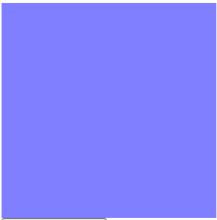
jQuery Examples

This is an existing paragraph. Appended text. Appended text. Appended text.

- Item 1
- Item 2
- New Item
- New Item
- New Item

Append Content

Animate Element



Change Color

Experiment-10

Develop a JavaScript program with Ajax (with HTML/CSS) for:

- a. Use ajax() method (without JQuery) to add the text content from the text file by sending ajax request.**
- b. Use ajax() method (with JQuery) to add the text content from the text file by sending ajax request.**
- c. Illustrate the use of getJSON() method in jQuery**
- d. Illustrate the use of parseJSON() method to display JSON values.**

Source Code:

A)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vanilla JS AJAX Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
    }
    #content {
      margin-top: 20px;
      padding: 10px;
      background-color: #f4f4f4;
      border-radius: 5px;
    }
  </style>
</head>
<body>
  <h1>Load Content from a File with AJAX (Vanilla JS)</h1>
  <button id="loadContent">Load Content</button>
  <div id="content"></div>

  <script>
    document.getElementById('loadContent').addEventListener('click', function() {
      var xhr = new XMLHttpRequest();
      xhr.open('GET', 'content.txt', true);

      xhr.onload = function() {
        if (this.status == 200) {
          document.getElementById('content').innerHTML = this.responseText;
        }
      }
    });
  </script>
</body>
</html>
```

```
    }  
};  
  
xhr.send();  
});  
</script>  
</body>  
</html>
```

Output:

B)

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>jQuery AJAX Example</title>  
  <style>  
    body {  
      font-family: Arial, sans-serif;  
    }  
    #content {  
      margin-top: 20px;  
      padding: 10px;  
      background-color: #e0f7fa;  
      border-radius: 5px;  
    }  
  </style>  
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>  
</head>  
<body>  
  <h1>Load Content from a File with AJAX (jQuery)</h1>  
  <button id="loadContent">Load Content</button>  
  <div id="content"></div>  
  
  <script>  
    $('#loadContent').on('click', function() {  
      $.ajax({  
        url: 'content.txt',  
        method: 'GET',
```

```
        success: function(data) {
            $('#content').html(data);
        },
        error: function() {
            $('#content').html('Error loading content.');
```

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

Output:

C)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>jQuery getJSON Example</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        #content {
            margin-top: 20px;
            padding: 10px;
            background-color: #f0f4c3;
            border-radius: 5px;
        }
    </style>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
    <h1>Load JSON Data with getJSON()</h1>
    <button id="loadContent">Load JSON</button>
    <div id="content"></div>

    <script>
        $('#loadContent').on('click', function() {
```

```
$.getJSON('data.json', function(data) {  
    var output = '<h2>' + data.title + '</h2>';  
    output += '<p>' + data.description + '</p>';  
    $('#content').html(output);  
});  
});  
</script>  
</body>  
</html>
```

data.json

```
{  
    "title": "JSON Example",  
    "description": "This is an example of loading JSON data using jQuery's getJSON() method."  
}
```

Output:

D)

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>parseJSON Example</title>  
    <style>  
        body {  
            font-family: Arial, sans-serif;  
        }  
        #content {  
            margin-top: 20px;  
            padding: 10px;  
            background-color: #f3e5f5;  
            border-radius: 5px;  
        }  
    </style>  
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>  
</head>  
<body>  
    <h1>Parsing JSON with parseJSON()</h1>
```



```
<button id="loadContent">Load and Parse JSON</button>
<div id="content"></div>

<script>
  $('#loadContent').on('click', function() {
    var jsonString = '{"name": "John Doe", "age": 30, "city": "New York"}';
    var parsedData = $.parseJSON(jsonString);

    var output = '<h2>Name: ' + parsedData.name + '</h2>';
    output += '<p>Age: ' + parsedData.age + '</p>';
    output += '<p>City: ' + parsedData.city + '</p>';
    $('#content').html(output);
  });
</script>
</body>
</html>
```

Output:

Viva Questions

1. What is the purpose of the `<title>` tag in an HTML document?

The `<title>` tag defines the title of the HTML document, displayed in the browser's title bar or tab. It's also used by search engines and when the page is bookmarked.

2. How does the `<marquee>` tag function, and what are its alternatives?

The `<marquee>` tag creates scrolling text or images. It's deprecated in HTML5 due to accessibility issues. Alternatives include using CSS animations or JavaScript to achieve similar effects.

3. Explain the difference between block-level and inline-level elements in HTML.

Block-level elements (e.g., `<div>`, `<p>`) take up the full width of the container and start on a new line.

Inline-level elements (e.g., ``, `<a>`) take up only as much width as their content and do not start on a new line.

4. What is the significance of the `<blockquote>` tag?

The `<blockquote>` tag is used for long quotations and typically indents the quoted text, distinguishing it from the rest of the content.

5. Can you differentiate between the `` and `` tags in HTML?

`` simply makes text bold with no added meaning.

`` not only makes text bold but also emphasizes its importance, providing semantic meaning for screen readers and search engines.

6. Why is the `<pre>` tag used, and how does it differ from the `<p>` tag?

The `<pre>` tag preserves whitespace, line breaks, and formatting as typed in the HTML document. It differs from the `<p>` tag, which collapses multiple spaces into a single space and doesn't preserve line breaks.

7. What are logical styles in HTML, and how do they enhance the presentation of text?

Logical styles (e.g., ``, ``) apply both stylistic formatting and semantic meaning. They help search engines and assistive technologies interpret the document structure and content more effectively.

8. How do you ensure that an HTML document is well-structured and accessible?

Use proper semantic tags (e.g., `<header>`, `<main>`, `<article>`), include alt attributes for images, use heading levels correctly, ensure proper nesting, and follow best practices for accessibility like using ARIA roles where needed.

9. **What is the purpose of using `rowspan` and `colspan` in an HTML table?**

`rowspan` merges cells vertically across multiple rows, and `colspan` merges cells horizontally across multiple columns. They are used to improve the table's layout, making it more readable by combining similar data points.

10. **Explain the significance of table headers (`<thead>`) and footers (`<tfoot>`) in an HTML table.**

`<thead>` defines the table's header section, typically used to label columns. `<tfoot>` defines the footer section, useful for summary rows or additional notes. These sections help organize data and improve table readability.

11. **What is the difference between an ID selector and a class selector in CSS?**

- An ID selector (`#id`) is unique and used for styling a single element. Each element can only have one ID, and an ID can only be used once per page.
- A class selector (`.class`) can be applied to multiple elements, allowing for reusable styles across different parts of the webpage.

12. **How does the element selector work, and when is it most appropriate to use it?**

The element selector (e.g., `p`, `h1`) targets all instances of a particular HTML element. It's most appropriate when you want to style every occurrence of a tag globally, like setting a universal font for all paragraphs (`p`).

13. **Can you explain the significance of using an external stylesheet instead of inline or internal CSS?**

An external stylesheet keeps CSS separate from HTML, promoting a cleaner, more maintainable structure. It allows for reuse across multiple HTML files and makes updating styles across a site easier, rather than embedding CSS in each document.

14. **What is a group selector in CSS, and how does it help in reducing redundancy?**

A group selector (e.g., `h1, h2, p`) applies the same styles to multiple elements, reducing redundancy in the code. Instead of repeating the same styles for each tag, grouping them makes the code more efficient and concise.

15. **Explain how attribute selectors work and provide a practical example of their use.**

Attribute selectors target elements based on attributes and their values. For example, `[type="text"]` selects input fields of type text:

EX: `input[type="text"] { background-color: #f0f0f0; }`

This applies styles only to specific input fields with the `type="text"` attribute.

16. How does the `hover` pseudo-class enhance the interactivity of links on a webpage?

The `:hover` pseudo-class enhances the interactivity of links by changing styles (like colour, background, etc.) when a user hovers over them. It's commonly used to indicate clickable elements and improve user experience:

Ex: `a:hover { text-decoration: underline; color: blue; }`

17. What are the advantages of using CSS to style HTML elements compared to inline styling?

CSS allows for separation of concerns (style from content), easier maintainability, and reuse across multiple elements or pages. It avoids inline clutter and provides flexibility through stylesheets, making it easier to update and manage designs globally.

18. How do you decide when to use an ID selector over a class selector?

Use an ID selector when styling a unique element on the page (e.g., a header, a specific section). Use a class selector for reusable styles across multiple elements (e.g., button styles, card designs).

19. What is the difference between `input[type="text"]` and `input[type="email"]`?

`input[type="text"]` accepts any text input, while `input[type="email"]` is designed to validate and accept email addresses.

20. Why is font size important in forms?

Font size ensures the readability of form elements. Proper sizing enhances user experience, making the form easier to fill out.

21. What is the purpose of the `placeholder` attribute in input fields?

The `placeholder` provides a hint inside the input field to guide the user on what type of information is expected (e.g., "Enter first name").

22. What is the purpose of using semantic elements like `<header>`, `<article>`, and `<aside>`?

Semantic elements define the structure and meaning of the content, making it more accessible for search engines and assistive technologies.

23. What is the use of the `<figure>` and `<figcaption>` tags?

The `<figure>` tag is used to encapsulate images, and `<figcaption>` provides a caption, making it easier to describe images or visual elements.

24. What is the purpose of the `<aside>` tag in this layout?

The `<aside>` tag holds content related to the main content, such as advertisements or additional information.

25. What is the purpose of `parseFloat` in this code?

`parseFloat` converts the input value (which is a string) into a floating-point number for mathematical operations.

26. Why is the switch-case structure used in this script?

The switch-case structure is used to handle multiple operations (like sum, product) based on the button the user clicks, making the code easier to manage.

27. What happens if you try to divide by zero in this calculator?

The program checks if the second number is zero before dividing and returns the message "Cannot divide by zero" to avoid mathematical errors.

28. How does the square root function work in this calculator?

It uses JavaScript's `Math.sqrt()` function to compute the square root of the first number.

29. Why are background and font colors applied in CSS rather than directly in HTML?

CSS separates content from design, making it easier to manage styles and ensuring the HTML structure focuses on content.

30. How does the `Math.pow` function work in JavaScript?

`Math.pow(num1, num2)` returns the value of `num1` raised to the power of `num2`.

31. What is the advantage of using `switch-case` instead of multiple `if-else` statements in this calculator?

`switch-case` is more readable and efficient when there are multiple distinct conditions like this, compared to chaining several `if-else` statements.

32. What is the purpose of the `file_exists()` function in PHP?

The `file_exists()` function is used to check if a file exists on the server before attempting to read or write to it. This helps to prevent errors when trying to access non-existent files.

33. How does the `file_get_contents()` function work in PHP?

`file_get_contents()` reads the entire content of a file into a string. For example it is used to retrieve the current visitor count from `counter.txt`.

34. What does the `file_put_contents()` function do?

`file_put_contents()` writes data to a file. It can either create a file if it doesn't exist or overwrite the contents of an existing file. In this program, it updates the visitor count in `counter.txt`.

35. How do you increment a value in PHP?

In PHP, you can increment a numeric value using the `++` operator. For example, `$current_count++` increases the value of `$current_count` by 1.

35. What is the role of the `.txt` file in this project?

The `.txt` file (`counter.txt`) is used to store the current number of visitors. It serves as persistent storage that PHP reads from and writes to, ensuring the visitor count is maintained between sessions.

36. How does PHP handle multiple users visiting the page simultaneously?

PHP handles multiple users by executing code on the server for each request independently. However, in the case of file handling (like updating `counter.txt`), race conditions may occur. Using file locking mechanisms or switching to database storage can prevent such issues.

37. What are the security concerns when writing to a file in PHP?

Some security concerns include:

- **File permission issues:** Incorrect permissions could allow unauthorized users to modify files.
- **Race conditions:** As mentioned, simultaneous access could corrupt data.
- **Injection attacks:** If user input is involved in file operations, it could lead to injection attacks like path traversal, which must be guarded against using proper validation.

38. Explain how you can handle click events in jQuery.

Answer: Click events in jQuery can be handled using the `.click()` method or the `.on('click', function() {})` method. These methods bind a function to the click event of a selected element, allowing for interactive user experiences.

39. What is the role of the `$(document).ready()` function?

Answer: The `$(document).ready()` function ensures that the DOM is fully loaded before executing any jQuery code. It helps prevent errors by ensuring that the elements are available for manipulation when the script runs.

40. What is AJAX, and how does it work?

Answer: AJAX (Asynchronous JavaScript and XML) is a technique used to send and receive data asynchronously without refreshing the entire web page. It works by using the `XMLHttpRequest` object or modern `fetch` API to send requests to the server, which then returns data (often in JSON format) that can be processed and displayed dynamically.

41. Explain the difference between the `XMLHttpRequest` and the `fetch` API.

Answer: `XMLHttpRequest` is an older API for making AJAX requests, while the `fetch` API is a modern alternative that is more powerful and flexible. `fetch` returns a Promise, making it easier to work with asynchronous code. Additionally, `fetch` supports more features, such as easier handling of response types.

42. How do you handle errors in AJAX requests?

Answer: Errors in AJAX requests can be handled using the `error` callback in jQuery's `$.ajax()` method or by using `.catch()` when working with Promises in the `fetch` API. It's essential to check the response status to determine if the request was successful.

43. What is the difference between `getJSON()` and `parseJSON()` in jQuery?

Answer: `getJSON()` is a shorthand method for making a GET request to fetch JSON data from a URL and automatically parses the response. `parseJSON()`, on the other hand, is used to convert a JSON string into a JavaScript object and is typically used when you have JSON data in string format.

44. What is the purpose of the `async` property in AJAX requests?

Answer: The `async` property determines whether the request is asynchronous or synchronous. When set to `true` (default), it allows the browser to continue processing other scripts while waiting for the response. If set to `false`, the browser will wait for the request to complete before continuing.

45. What happens if an AJAX request is made to a URL that does not exist?

Answer: If an AJAX request is made to a non-existent URL, the server will typically respond with a 404 Not Found status. In the error callback or `.catch()` block, you can handle this response and inform the user that the requested resource is not available.

46. What is the difference between the `fadeIn()` and `animate()` methods?

Answer: The `fadeIn()` method specifically fades an element into view by changing its opacity, while the `animate()` method can change any CSS properties, including size, position, and color, providing more flexibility in animations.