Internet And Web Programming

IS2001-1

Dr. Jason Elroy Martis

Lab Programs for reference and Familiarity

1. **Using HTML5 performing following tasks:**

i) **Adding an image.**

ii) **Link to different pages and same pages**

iii) **Redirecting current page to another page after 5 seconds using HTML5 meta tag**

Adding an Image using HTML5:

HTML Code:

```
html
<!DOCTYPE html>
<html>
<body>
    <img src="image_url.jpg" alt="Description of the image">
</body>
</html>
```

Replace `image_url.jpg` with the actual URL or path of your image.

Creating Links to Different Pages and the Same Page using HTML5:

HTML Code:

```
html
<!DOCTYPE html>
<html>
<body>
    <!-- Link to a different page -->
    <a href="http://example.com">Visit Example.com</a>

    <!-- Link to a section on the same page -->
    <a href="#section1">Go to Section 1</a>

    <!-- Section to link to -->
    <h2 id="section1">Section 1</h2>
    <p>Content of Section 1...</p>
</body>
</html>
```

Replace `http://example.com` with the URL you want to link to. The `#section1` is an anchor link that scrolls the page to the element with the `id="section1"`.

Redirecting a Page After 5 Seconds using HTML5:

HTML Code:

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="refresh" content="5; url=http://example.com">
```

```
</head>
<body>
    <p>The page will redirect in 5 seconds...</p>
</body>
</html>
```

The `meta` tag in the `head` section of the HTML document sets the page to redirect after 5 seconds to `http://example.com`. Change the URL to the desired destination.

These HTML snippets demonstrate the basic implementation of each task. Remember to replace URLs and paths with those relevant to your specific use case.

2. Demonstrate the following HTML5 Semantic tags – < article >, < aside >, < details >, < figcaption >, < figure >, < footer >, < header >, < main >, < mark >, < section > for a webpage that gives information about travel experience.

```
<!DOCTYPE html>
<html>
<head>
    <title>Travel Experience</title>
</head>
<body>

<header>
    <h1>My Amazing Travel Journey</h1>
    <nav>
        <!-- Navigation links can go here -->
    </nav>
</header>

<main>
    <article>
        <header>
            <h2>A Weekend in Paris</h2>
            <p>Published on: <time datetime="2023-01-15">January 15,
2023</time></p>
        </header>
        <section>
            <h3>The Heart of Paris</h3>
            <p>My journey began in the heart of Paris, where I explored the
iconic Eiffel Tower...</p>
            <figure>
                <img src="eiffel_tower.jpg" alt="Eiffel Tower">
                <figcaption>The majestic Eiffel Tower.</figcaption>
            </figure>
        </section>
        <section>
            <h3>Local Cuisine</h3>
            <p>I also had the chance to enjoy the local cuisine, especially
the delightful croissants...</p>
        </section>
        <footer>
            <p>Author: Jane Doe</p>
        </footer>
    </article>

    <aside>
```

```
        <h4>Related Trips</h4>
        <p>Check out my journey through the Italian countryside...</p>
        <!-- Links to related articles -->
    </aside>
</main>

<footer>
    <p>© 2024 Travel Blog. All rights reserved.</p>
</footer>

</body>
</html>
```

Explanation of tags

- `<article>`: Encapsulates the main content of the travel story.
- `<aside>`: Contains related but tangential information, like links to other travel stories.
- `<details>`: Not used in this example, but it could be used for additional details like travel tips.
- `<figcaption>`: Provides a caption for the image inside the `<figure>` element.
- `<figure>`: Wraps an image and its caption.
- `<footer>`: Footer of the article and the webpage.
- `<header>`: Contains introductory content for both the web page and individual articles.
- `<main>`: Represents the main content of the webpage.
- `<mark>`: Not used in this example, but it could highlight parts of the text.
- `<section>`: Defines sections within an article.

Remember, this is just a basic structure and can be expanded with CSS for styling and more HTML for additional content.

My Amazing Travel Journey

A Weekend in Paris

Published on: January 15, 2023

The Heart of Paris

My journey began in the heart of Paris, where I explored the iconic Eiffel Tower...

The majestic Eiffel Tower.

3. Modify the `<li>` tag in an HTML document to include specific styling properties:

- **Apply a margin of 5px.**
- **Set the padding to 10px at the top, 20px on the right, 10px at the bottom, and 20px on the left.**
- **Additionally, demonstrate how to use custom image logos as list-style types for these list items**

Also provide the code to achieve these styling changes.

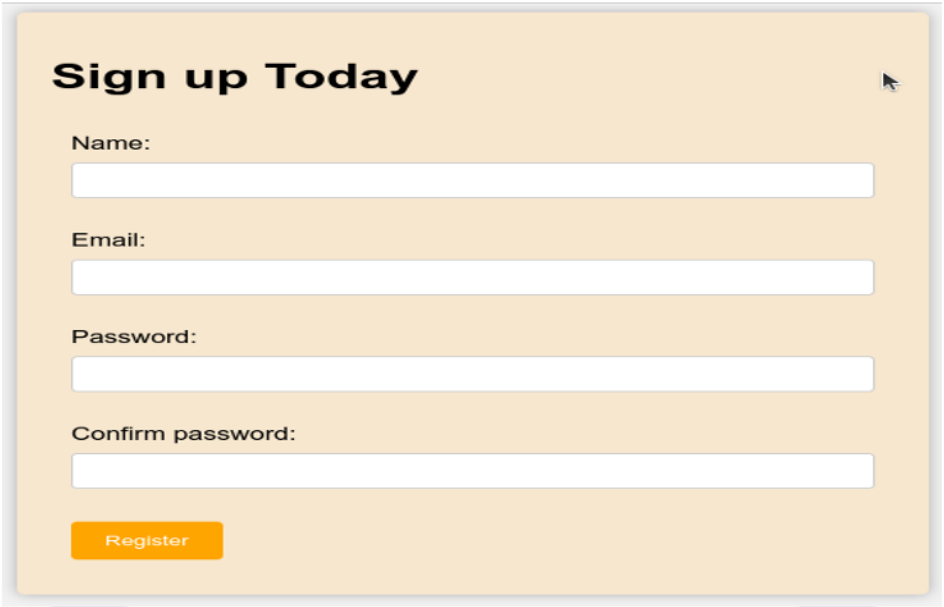
```
<!DOCTYPE html>
<html>
<head>
<style>
    ul.custom-list li {
        margin: 5px;
        padding: 10px 20px 10px 20px;
        list-style-type: none; /* This removes the default list-style */
        background-image: url('path_to_your_logo_image.jpg'); /* Replace
with your logo image path */
        background-repeat: no-repeat;
        background-position: left center; /* Adjust as needed */
        text-indent: 40px; /* Adjust based on the size of your logo image
*/
    }
</style>
</head>
<body>

<h2>Custom Styled List</h2>
<ul class="custom-list">
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>List Item 3</li>
    <!-- Add more list items as needed -->
</ul>

</body>
</html>
```

In this code:

- The CSS under the `<style>` tag is used to style the list items (`<li>` elements) inside a list with the class `custom-list`.
- The `margin` and `padding` are set according to your specifications.
- The `list-style-type` is set to `none` to remove the default bullet points.
- The `background-image` is used to set a custom image as the bullet point. Replace `'path_to_your_logo_image.jpg'` with the actual path to your image.
- The `text-indent` property is used to align the text properly with the custom image. You might need to adjust this value based on the size of your image.

Remember to replace the placeholder path in `url('path_to_your_logo_image.jpg')` with the actual path to your custom logo image.

4. Create the following web page using HTML and CSS with tabular layout.




```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Sign Up Page</title>
<style>
  .signup-container {
    width: 300px;
    margin: 50px auto;
    padding: 20px;
    background-color: #f7f7f7;
    border: 1px solid #ebebeb;
    border-radius: 8px;
  }

  table {
    width: 100%;
  }

  td {
    padding: 8px;
  }

  input[type="text"],
  input[type="email"],
  input[type="password"] {
    width: 94%;
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
  }

  input[type="submit"] {
    width: 100%;
```

```
    padding: 10px;
    border: none;
    border-radius: 4px;
    background-color: #ffa500;
    color: white;
    cursor: pointer;
  }

  input[type="submit"]:hover {
    background-color: #e69500;
  }

  h2 {
    text-align: center;
    color: #333;
  }
</style>
</head>
<body>

<div class="signup-container">
  <h2>Sign up Today</h2>
  <form action="/submit-your-form" method="post">
    <table>
      <tr>
        <td>Name:</td>
        <td><input type="text" name="name" required></td>
      </tr>
      <tr>
        <td>Email:</td>
        <td><input type="email" name="email" required></td>
      </tr>
      <tr>
        <td>Password:</td>
        <td><input type="password" name="password" required></td>
      </tr>
      <tr>
        <td>Confirm password:</td>
        <td><input type="password" name="confirm-password" required></td>
      </tr>
      <tr>
        <td colspan="2"><input type="submit" value="Register"></td>
      </tr>
    </table>
  </form>
</div>

</body>
</html>
```

This code sets up a form inside a div container with a class `signup-container`. The CSS styles are used to center this container on the page, set the background color, and style the input fields and button. The form itself is laid out using a table with rows for each input field.

Remember to replace the `action` attribute in the `<form>` tag with the URL where you want to send the form data when the 'Register' button is clicked. If you're going to test the form's submission, you'll need a server-side script to handle the POST request.

5. Develop a Javascript program to get CGPA as input and print the percentage equivalance of CGPA.

To convert CGPA to a percentage, you would typically use a formula. Commonly, in many education systems, the formula is as simple as multiplying the CGPA by a number (often 9.5 in the Indian grading system, for example). However, this can vary depending on the specific academic regulations.

Here's a simple HTML and JavaScript program that takes CGPA as input and converts it to a percentage using the formula `Percentage = CGPA * 9.5`. You can change the multiplier as per the specific conversion rule of your academic institution.

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

<script>
function convertCGPA() {
    var cgpa = document.getElementById('cgpa').value;
    var multiplier = 9.5; // Change this multiplier as per your academic
regulations
    var percentage = cgpa * multiplier;
    document.getElementById('percentage').innerText     =     "Percentage
Equivalent: " + percentage.toFixed(2) + "%";
}
</script>

</head>
<body>

<h2>CGPA to Percentage Converter</h2>

<label for="cgpa">Enter your CGPA:</label>
<input type="number" id="cgpa" min="0" max="10" step="0.01" required>
<button onclick="convertCGPA()">Convert</button>

<p id="percentage"></p>

</body>
</html>
```

When you enter a CGPA into the input box and click "Convert", the JavaScript function `convertCGPA` is called. It retrieves the value, applies the conversion, and then displays the percentage equivalent on the page.

Remember to validate and adapt the multiplier according to your specific needs. The given script assumes a simple conversion which might not apply to all CGPA systems.

6. Design an Exam Registration Form (NUCAT) using HTML with details like Name, Email ID, Mobile, Exam mode, Fees paid and Display the Exam details based on the fees paid using Javascript.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>NUCAT Exam Registration Form</title>
<script>
function displayExamDetails() {
    var name = document.getElementById('name').value;
    var email = document.getElementById('email').value;
    var mobile = document.getElementById('mobile').value;
    var examMode = document.querySelector('input[name="exam-
mode"]:checked').value;
    var feesPaid = document.getElementById('fees-paid').value;

    var examDetails = "Registration Details:\n";
    examDetails += "Name: " + name + "\n";
    examDetails += "Email ID: " + email + "\n";
    examDetails += "Mobile: " + mobile + "\n";
    examDetails += "Exam Mode: " + examMode + "\n";

    // Assume certain fee thresholds for different exam details
    if (feesPaid >= 100) {
        examDetails += "Exam Details: You are registered for the Standard
NUCAT Exam.\n";
    } else {
        examDetails += "Exam Details: Fees paid are insufficient for
registration.\n";
    }

    alert(examDetails);
}
</script>
</head>
<body>

<h2>NUCAT Exam Registration Form</h2>

<form onsubmit="event.preventDefault(); displayExamDetails();">
    <label for="name">Name:</label>
    <input type="text" id="name" required><br><br>

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

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

    <span>Exam Mode:</span><br>
    <input type="radio" id="online" name="exam-mode" value="Online"
checked>
    <label for="online">Online</label><br>
    <input type="radio" id="offline" name="exam-mode" value="Offline">
    <label for="offline">Offline</label><br><br>

    <label for="fees-paid">Fees Paid:</label>
```

```
    <input type="number" id="fees-paid" required><br><br>

    <input type="submit" value="Register">
</form>

</body>
</html>
```

In this code, we have:

- An HTML form for the candidate to enter their details.
- Radio buttons to select the exam mode (Online/Offline).
- A JavaScript function `displayExamDetails` that is called when the form is submitted. It gathers the form data and displays an alert with the exam details based on the fees paid.

You should adjust the fee thresholds and exam details in the JavaScript function to match the actual conditions of the NUCAT exam.

Please note, the `onsubmit` event handler on the form is calling `event.preventDefault()` to prevent the form from actually submitting and refreshing the page. Instead, it calls the `displayExamDetails` function to process the form data and display the results.

7. Write JavaScript to validate the following fields of the Registration form.
i) Name (field can not be empty)
ii) Age (Number validation)
iii) Password
iv) Confirm Password

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Registration Form Validation</title>
<script>
function validateRegistrationForm() {
    var name = document.getElementById('name').value;
    var age = document.getElementById('age').value;
    var password = document.getElementById('password').value;
    var confirmPassword = document.getElementById('confirm-
password').value;

    // Name validation
    if (name.trim() === '') {
        alert('Name field cannot be empty.');
        return false;
    }

    // Age validation (must be a number)
    if (!age || isNaN(age)) {
        alert('Please enter a valid age.');
        return false;
    }
```

```
    // Password validation
    if (password.trim() === '') {
        alert('Password field cannot be empty.');
        return false;
    }

    // Confirm Password validation
    if (password !== confirmPassword) {
        alert('Passwords do not match.');
        return false;
    }

    alert('Registration form is valid!');
    return true; // Form is valid (here you would typically submit the form
to your server)
}
</script>
</head>
<body>

<h2>Registration Form</h2>

<form onsubmit="return validateRegistrationForm();">
    <label for="name">Name:</label>
    <input type="text" id="name" required><br><br>

    <label for="age">Age:</label>
    <input type="text" id="age" required><br><br>

    <label for="password">Password:</label>
    <input type="password" id="password" required><br><br>

    <label for="confirm-password">Confirm Password:</label>
    <input type="password" id="confirm-password" required><br><br>

    <input type="submit" value="Register">
</form>

</body>
</html>
```

In this script:

- The `validateRegistrationForm` function is called when the form is submitted.
- The function checks if the `name` field is empty.
- It validates whether the `age` is a number using `isNaN`.
- It checks if the `password` field is empty.
- It verifies that the `password` and `confirmPassword` fields match.
- If any validation fails, it will alert the user and return `false`, preventing form submission.
- If all validations pass, it alerts that the form is valid and would return `true`. In a real-world scenario, you would typically allow the form to be submitted by returning `true` or you might handle the submission process via JavaScript.

8. Write a JavaScript to design a simple calculator to perform the following operations: sum, product, difference and quotient.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Simple Calculator</title>
<script>
function calculate(operation) {
    var num1 = parseFloat(document.getElementById('num1').value);
    var num2 = parseFloat(document.getElementById('num2').value);
    var result;

    switch (operation) {
        case 'add':
            result = num1 + num2;
            break;
        case 'subtract':
            result = num1 - num2;
            break;
        case 'multiply':
            result = num1 * num2;
            break;
        case 'divide':
            if (num2 === 0) {
                alert('Cannot divide by zero.');
                return;
            }
            result = num1 / num2;
            break;
        default:
            result = 'Invalid operation';
    }

    document.getElementById('result').value = result;
}
</script>
</head>
<body>

<h2>Simple Calculator</h2>

<label for="num1">Number 1:</label>
<input type="number" id="num1" required><br><br>

<label for="num2">Number 2:</label>
<input type="number" id="num2" required><br><br>

<button onclick="calculate('add')">Sum</button>
<button onclick="calculate('subtract')">Difference</button>
<button onclick="calculate('multiply')">Product</button>
<button onclick="calculate('divide')">Quotient</button><br><br>

<label for="result">Result:</label>
<input type="text" id="result" readonly><br><br>

</body>
</html>
```

Here's what the code does:

- HTML defines a simple form with two numeric input fields for the user to enter numbers.
- It provides four buttons for the user to choose the operation: sum, product, difference, and quotient.
- The JavaScript function `calculate` takes an `operation` parameter that determines which arithmetic operation to perform.
- It reads the values from the input fields, performs the requested operation, and outputs the result in the 'result' field.
- The `parseFloat` function is used to convert the input values to floating-point numbers, ensuring that arithmetic operations are performed correctly even with decimal numbers.
- Division by zero is checked to prevent errors.
- The result is displayed in a read-only input field.

9. Write a PHP code to Connect to Database, Create a table College (Dname, Dno,Number_Of_faculty) Insert 3 records of your choice, select Dno with max faculty

```
<?php
$servername = "localhost"; // Your database server, e.g., localhost
$username = "username"; // Your database username
$password = "password"; // Your database password
$dbname = "your_db_name"; // Your database name

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// SQL to create table
$sqlCreateTable = "CREATE TABLE College (
  Dname VARCHAR(50),
  Dno INT PRIMARY KEY,
  Number_Of_faculty INT
)";

if ($conn->query($sqlCreateTable) === TRUE) {
  echo "Table College created successfully";
} else {
  echo "Error creating table: " . $conn->error;
}

// Insert records
$sqlInsert = "INSERT INTO College (Dname, Dno, Number_Of_faculty) VALUES
('Computer Science', 101, 25),
('Mechanical Engineering', 102, 15),
('Electronics', 103, 20)";

if ($conn->query($sqlInsert) === TRUE) {
  echo "New records created successfully";
} else {
  echo "Error: " . $sqlInsert . "<br>" . $conn->error;
}
```

```
// Select Dno with max faculty
$sqlSelect = "SELECT Dno FROM College ORDER BY Number_Of_faculty DESC LIMIT
1";

$result = $conn->query($sqlSelect);

if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "Dno with max faculty: " . $row["Dno"];
  }
} else {
  echo "0 results";
}

$conn->close();
?>
```

Important Notes:

- Replace `localhost`, `username`, `password`, and `your_db_name` with your actual database server name, username, password, and database name.
- The script assumes your MySQL server is up and running and PHP is properly configured to connect to it.
- This script doesn't include error handling for the database connection and queries beyond basic checks. For production environments, you should handle errors more gracefully and securely.
- Always sanitize and validate your inputs when dealing with databases to prevent SQL injection and other security vulnerabilities.
- The script does not include any form of user authentication or validation, which is crucial for a secure application.
- The `CREATE TABLE` command will fail if a table named `College` already exists. You may need to check for the table's existence or handle the error differently depending on your needs.

10. Write a PHP program to connect to the database. Create a table named "Student" with following fields (sno, sname, percentage). Insert 3 records of your choice. Delete a particular record. Use Update and Select query to show the table details after deletion.

```
<?php
$servername = "localhost"; // Your database server
$username = "username"; // Your database username
$password = "password"; // Your database password
$dbname = "your_db_name"; // Your database name

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// SQL to create table
```

```
$sqlCreateTable = "CREATE TABLE Student (
    sno INT PRIMARY KEY,
    sname VARCHAR(50),
    percentage DECIMAL(5,2)
)";

if ($conn->query($sqlCreateTable) === TRUE) {
    echo "Table Student created successfully<br>";
} else {
    echo "Error creating table: " . $conn->error . "<br>";
}

// Insert records
$sqlInsert = "INSERT INTO Student (sno, sname, percentage) VALUES
(1, 'John Doe', 87.5),
(2, 'Jane Smith', 93.0),
(3, 'Emily Johnson', 78.2)";

if ($conn->query($sqlInsert) === TRUE) {
    echo "New records created successfully<br>";
} else {
    echo "Error: " . $sqlInsert . "<br>" . $conn->error . "<br>";
}

// Delete a specific record
$sqlDelete = "DELETE FROM Student WHERE sno = 2"; // Change the sno
value as needed

if ($conn->query($sqlDelete) === TRUE) {
    echo "Record deleted successfully<br>";
} else {
    echo "Error deleting record: " . $conn->error . "<br>";
}

// Update a record
$sqlUpdate = "UPDATE Student SET percentage = 82.5 WHERE sno = 3";

if ($conn->query($sqlUpdate) === TRUE) {
    echo "Record updated successfully<br>";
} else {
    echo "Error updating record: " . $conn->error . "<br>";
}

// Select and display all records
$sqlSelect = "SELECT sno, sname, percentage FROM Student";
$result = $conn->query($sqlSelect);

if ($result->num_rows > 0) {
    echo "Updated table details:<br>";
    while($row = $result->fetch_assoc()) {
        echo "sno: " . $row["sno"]. " - Name: " . $row["sname"]. " -
Percentage: " . $row["percentage"]. "<br>";
    }
} else {
    echo "0 results<br>";
}

$conn->close();
?>
```

Before running this script, replace `localhost`, `username`, `password`, and `your_db_name` with your actual database host, username, password, and the name of your database.

This script is very straightforward. However, keep in mind that in a production environment, you would need to take additional security measures such as prepared statements to prevent SQL injection, and you would likely be interacting with the database in a more modular and object-oriented manner.

11. Write a React JS code to demonstrate how to locally manage a component's state

In React, a component's state can be managed locally using the `useState` hook for functional components or `this.state` and `this.setState` for class components. Below, I will provide an example for each case.

Functional Component with `useState` Hook

Here's a simple functional component that uses the `useState` hook to manage its state:

```
jsx
import React, { useState } from 'react';

function Counter() {
  // Initialize the state with useState hook
  const [count, setCount] = useState(0);

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increase</button>
      <button onClick={() => setCount(count - 1)}>Decrease</button>
    </div>
  );
}

export default Counter;
```

In this example, `useState` is used to create a state variable `count` along with a function `setCount` that can update this state. Clicking on the "Increase" button increases the count by 1, while clicking on the "Decrease" button decreases the count by 1.

Class Component with State

Here's how you can manage state in a class component:

```
jsx
import React from 'react';

class Counter extends React.Component {
  constructor(props) {
    super(props);
    // Initialize the state in the constructor
    this.state = { count: 0 };
  }
```

```
  increaseCount = () => {
    this.setState({ count: this.state.count + 1 });
  };

  decreaseCount = () => {
    this.setState({ count: this.state.count - 1 });
  };

  render() {
    return (
      <div>
        <h1>Count: {this.state.count}</h1>
        <button onClick={this.increaseCount}>Increase</button>
        <button onClick={this.decreaseCount}>Decrease</button>
      </div>
    );
  }
}

export default Counter;
```

In this class component example, the state is initialized in the constructor, and `this.setState` is used to update the state. Methods `increaseCount` and `decreaseCount` are defined to handle the logic for increasing and decreasing the count, respectively.

Both examples show how to manage the state locally within a component, which is essential for handling user inputs, events, and other dynamic data within the component.

12. Create a Web Server using HTTP Module (Node JS)

To create a simple web server using the HTTP module in Node.js, you can follow the steps below. This server will listen on port 3000 and respond with a simple message to any request.

First, make sure you have Node.js installed on your system. Then, you can create a file named `server.js` and add the following code:

```
javascript
const http = require('http');

// The port on which the server will listen
const port = 3000;

// Create a server object
const server = http.createServer((req, res) => {
  // Set the response HTTP header with HTTP status and Content type
  res.writeHead(200, {'Content-Type': 'text/plain'});

  // Send the response body "Hello World"
  res.end('Hello World\n');
});

// The server starts listening on port 3000
server.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

To run your server, open your terminal, navigate to the directory where `server.js` is located, and run the command:

```
bash
node server.js
```

The console should output `Server running at http://localhost:3000/`, indicating that the server is running and listening for requests on port 3000. You can open a web browser and navigate to `http://localhost:3000/` to see the server's response.

This basic example does not handle different URLs or HTTP methods. It simply returns "Hello World" for every request. In a real-world application, you would use frameworks like Express.js to handle routing and middleware more effectively.

13. Demonstrate Inline Styling (color, backgroundColor, margin, font-weight,marginBottom) with JSX in React JS.

In React, inline styles are specified as objects with camelCased properties rather than a string like in regular HTML. Here's a simple demonstration of inline styling in a React functional component:

```
jsx
import React from 'react';

function InlineStylingComponent() {
  // Define the styles object
  const styleObj = {
    color: 'blue', // Sets the text color
    backgroundColor: 'lightgray', // Sets the background color
    margin: '10px', // Sets the margin on all sides
    fontWeight: 'bold', // Makes the font bold
    marginBottom: '20px' // Sets the bottom margin
  };

  return (
    <div style={styleObj}> {/* Apply the styles using the style attribute
*/}
      This text demonstrates inline styling in React.
    </div>
  );
}

export default InlineStylingComponent;
```

In this example:

- A `styleObj` object is created, containing all the inline styles you want to apply.
- Styles are written in camelCase instead of kebab-case (e.g., `marginBottom` instead of `margin-bottom`).
- The `style` attribute is then used on the JSX element, and `styleObj` is passed to it.

When you render `InlineStylingComponent`, you'll see the text styled according to the properties defined in `styleObj`. Inline styling is straightforward for applying specific styles directly to elements, but it's generally recommended to use CSS classes for larger, more complex applications for better separation of concerns and maintainability.

Happy Coding!!!

😊😊😊