fwed lab ans

1. Create an HTML form that includes:
   a. Text input, email input, password field, dropdown, radio buttons, and a submit button.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple HTML Form</title>
</head>
<body>

  <h2>Registration Form</h2>

  <form>

    <!-- Text Input -->
    <label>Full Name:</label><br>
    <input type="text" name="fullname" placeholder="Enter your name">
    <br><br>

    <!-- Email Input -->
    <label>Email:</label><br>
    <input type="email" name="email" placeholder="Enter your email">
    <br><br>

    <!-- Password Field -->
    <label>Password:</label><br>
    <input type="password" name="password">
    <br><br>

    <!-- Dropdown -->
    <label>Course:</label><br>
    <select name="course">
```

```
        <option value="">Select a course</option>
        <option value="cse">Computer Science</option>
        <option value="ece">Electronics</option>
        <option value="mech">Mechanical</option>
    </select>
    <br><br>

    <!-- Radio Buttons -->
    <label>Gender:</label><br>
    <input type="radio" name="gender" value="male"> Male<br>
    <input type="radio" name="gender" value="female"> Female<br>
    <input type="radio" name="gender" value="other"> Other
    <br><br>

    <!-- Submit Button -->
    <input type="submit" value="Submit">

  </form>

</body>
</html>
```

---

2. Style a navigation bar using CSS Flexbox or Grid to make it fully responsive (mobile and desktop views).

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Responsive Navbar</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

  <nav class="navbar">
```

```
      <div class="logo">MySite</div>

      <ul class="nav-links">
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Services</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>

</body>
</html>
```

```
/* Reset */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Navbar container */
.navbar {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background-color: #333;
    padding: 15px 20px;
}

/* Logo */
.logo {
    color: white;
    font-size: 20px;
    font-weight: bold;
}

/* Navigation links */
.nav-links {
```

```
    list-style: none;
    display: flex;
}

.nav-links li {
    margin-left: 20px;
}

.nav-links a {
    text-decoration: none;
    color: white;
    font-size: 16px;
}

/* Hover effect */
.nav-links a:hover {
    color: #00bcd4;
}

/* 📱 Mobile view */
@media (max-width: 600px) {
    .navbar {
        flex-direction: column;
        align-items: flex-start;
    }

    .nav-links {
        flex-direction: column;
        width: 100%;
        margin-top: 10px;
    }

    .nav-links li {
        margin: 10px 0;
    }
}
```

Given an array of objects representing students
with name and grade, write a JavaScript function to:
a. Filter students with grades above 90.
b. Return an array of their names in uppercase.

```
const students = [
   { name: "Rohan", grade: 95 },
   { name: "Anita", grade: 88 },
   { name: "Rahul", grade: 92 },
   { name: "Neha", grade: 85 }
];

function getTopStudents(students) {
   return students
      .filter(student => student.grade > 90)
      .map(student => student.name.toUpperCase());
}

const result = getTopStudents(students);
console.log(result);
```

```
["ROHAN", "RAHUL"]
```

`map()` is a **JavaScript array method** that:

- **loops through each element** of an array
- **changes each element**
- **returns a NEW array**
- **does NOT modify the original array**

`filter()` is a **JavaScript array method** that:

- goes through **each element** of an array
- **checks a condition** (true / false)

- **keeps only the elements** that return `true`
- returns a **NEW array**
- does **not** change the original array

Build a simple calculator in JavaScript that can add, subtract, multiply, and divide. The interface should have buttons for digits and operations, and display the result.

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

  <style>
    body {
      display: flex;
      justify-content: center;
      margin-top: 50px;
      font-family: Arial, sans-serif;
    }

    .calculator {
      border: 2px solid #333;
      padding: 15px;
      width: 220px;
    }

    #display {
      width: 100%;
      height: 40px;
      margin-bottom: 10px;
      font-size: 18px;
      text-align: right;
```

```
        padding-right: 5px;
    }

    button {
        width: 45px;
        height: 40px;
        margin: 3px;
        font-size: 16px;
        cursor: pointer;
    }

    .operator {
        background-color: #ddd;
    }
  </style>
</head>
<body>

  <div class="calculator">
    <input type="text" id="display" disabled>

    <br>

    <button onclick="clearDisplay()">C</button>
    <button onclick="appendValue('/')">/</button>
    <button onclick="appendValue('*')">*</button>
    <button onclick="appendValue('-')">-</button>

    <br>

    <button onclick="appendValue('7')">7</button>
    <button onclick="appendValue('8')">8</button>
    <button onclick="appendValue('9')">9</button>
    <button onclick="appendValue('+')">+</button>

    <br>

    <button onclick="appendValue('4')">4</button>
```

```
        <button onclick="appendValue('5')">5</button>
        <button onclick="appendValue('6')">6</button>
        <button onclick="calculate()">=</button>

        <br>

        <button onclick="appendValue('1')">1</button>
        <button onclick="appendValue('2')">2</button>
        <button onclick="appendValue('3')">3</button>

        <br>

        <button onclick="appendValue('0')">0</button>
    </div>

    <script>
        let display = document.getElementById("display");

        function appendValue(value) {
            display.value += value;
        }

        function clearDisplay() {
            display.value = "";
        }

        function calculate() {
            try {
                display.value = eval(display.value);
            } catch {
                display.value = "Error";
            }
        }
    </script>

</body>
</html>
```

🧠 How it works (simple explanation)

- **Display** → `<input>` shows numbers and result
- **Buttons** → call JS functions using `onclick`
- `appendValue()` → adds clicked number/operator
- `clearDisplay()` → clears screen
- `calculate()` → uses `eval()` to compute result

Use the Fetch API to get data from a public API (like JSONPlaceholder) and display the list of posts in the DOM.

Fetch API Example: Display Posts in the DOM

📌 What this does

- Fetches posts from **JSONPlaceholder**
- Displays **post title and body** on the webpage
- Uses **Fetch API +** `.then()`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Fetch Posts</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
    }
    .post {
      border: 1px solid #ccc;
      padding: 10px;
```

```
        margin-bottom: 10px;
      }
      h3 {
        margin: 0;
      }
    </style>
</head>
<body>

    <h2>Posts</h2>
    <div id="posts"></div>

    <script>
      fetch("https://jsonplaceholder.typicode.com/posts")
        .then(response => response.json())
        .then(data => {
          const postsDiv = document.getElementById("posts");

          data.forEach(post => {
            const postElement = document.createElement("div");
            postElement.className = "post";

            postElement.innerHTML = `
              <h3>${post.title}</h3>
              <p>${post.body}</p>
            `;

            postsDiv.appendChild(postElement);
          });
        })
        .catch(error => {
          console.log("Error fetching data:", error);
        });
    </script>

</body>
</html>
```

🧠 Step-by-step explanation (important)

1. `fetch(url)`

   → Sends a request to the API

2. `response.json()`

   → Converts response to JavaScript object

3. `data.forEach()`

   → Loops through all posts

4. `document.createElement()`

   → Creates HTML elements dynamically

5. `appendChild()`

   → Adds posts to the DOM

Create a React component that accepts a user prop (object with name, email, age) and displays it in a card format.

1. **Name two semantic HTML tags and explain their purpose.**

- `<header>`: Used to contain introductory content, typically including headings, logos, or navigation links.
- `<article>`: Represents a self-contained composition in a document, such as a blog post, a news story, or a comment.

1. **What is the purpose of the `alt` attribute in an HTML `<img>` tag?**
   The `alt` (alternate text) attribute provides a text description of the image for screen readers (improving accessibility) and displays text if the image fails to load (improving user experience and SEO).
2. **What is the purpose of the HTML `<form>` element?**
   The `<form>` element is a container used to collect user input. It defines the area for interactive controls (like inputs, buttons, etc.) and specifies how

and where the data collected from these controls should be submitted (e.g., via `action` and `method` attributes).

3. **What are semantic HTML elements? Provide examples and explain their importance for accessibility and SEO.**
   Semantic HTML elements are tags that clearly define the purpose and meaning of the content they contain, rather than just dictating how the content should look.

- **Examples**: `<nav>`, `<main>`, `<section>`, `<aside>`, `<footer>`, `<figure>`.
- **Importance**:
- **Accessibility**: They help assistive technologies (like screen readers) interpret the structure and meaning of the page, making it usable for people with disabilities.
- **SEO**: Search engines use semantic elements to better understand the hierarchy and importance of the content, which can improve ranking.

CSS

1. **Name the four parts of the CSS Box Model from inside to outside and briefly explain?**
   The four parts of the CSS Box Model, from innermost to outermost, are:
2. **Content**: The actual content of the element (e.g., text, image). Its dimensions are set by `width` and `height`.
3. **Padding**: Transparent space surrounding the content, inside the border. It helps create visual breathing room.
4. **Border**: A visible line that wraps the padding and content.
5. **Margin**: Transparent space outside the border, used to separate the element from other elements.
6. **What are the three ways to include CSS in an HTML document?**
7. **External Stylesheets**: Linking a separate `.css` file using the `<link>` tag in the `<head>`. (Most common and recommended.)
8. **Internal/Embedded Styles**: Placing CSS rules inside a `<style>` tag within the `<head>` of the HTML document.
9. **Inline Styles**: Applying CSS rules directly to an element using the `style` attribute.

10. **What is the box model in CSS? Explain its components with a diagram.***See Theoretical Question 5 for explanation of components (Content, Padding, Border, Margin).*
    The Box Model is a conceptual structure used when rendering HTML elements. Every HTML element is treated as a rectangular box defined by these four layers.
11. **What are CSS BEM methodology and CSS Modules? How do they solve CSS scoping issues?**

- **BEM (Block, Element, Modifier)**: A naming convention for CSS classes (e.g., `card`, `card__title`, `card--large`). It provides structure and makes styles predictable, solving scoping issues by creating unique, descriptive class names that are unlikely to conflict.
- **CSS Modules**: A system where CSS files are locally scoped by default. When imported into a JavaScript component (like in React), the class names are automatically transformed into unique hashes (e.g., `App_header_ad832f`). This technologically guarantees that styles defined in one module will not affect others.

CSS Layout Comparison

1. **Compare and contrast Flexbox and CSS Grid layout systems. When would you choose one over the other?**
Feature	CSS Flexbox	CSS Grid
Primary Use	One-dimensional layouts (rows *or* columns)	Two-dimensional layouts (rows *and* columns simultaneously)
Control	Content-out (based on item size)	Layout-in (based on explicit tracks/cells)
Best For	Navigation bars, distributing space between items, component alignment	Overall page structure, complex gallery layouts

- **Choose Flexbox**: For aligning items along a single axis (e.g., centering text, spacing out navigation links).
- **Choose Grid**: For large-scale page structure or any design that requires items to align in both horizontal and vertical dimensions (rows and columns).

1. **Compare the primary use cases for CSS Flexbox versus CSS Grid.**

- **Flexbox Use Cases**: Component-level alignment, navigation bars, vertical centering, and arranging items when the exact number of rows/columns is less important than item spacing and alignment.
- **CSS Grid Use Cases**: Page templates, main layout definition (header, sidebar, main content, footer), and creating complex multi-column/multi-row interfaces.

1. **What does responsive web design mean?**
   Responsive web design (RWD) is an approach to web development that ensures the layout, content, and functionality of a website adapt and respond fluidly to the screen size and device capabilities (desktop, tablet, mobile) of the user.

JavaScript Theory

1. **What is the difference between synchronous and asynchronous code in JavaScript? Explain with examples.**
2. **Explain the difference between synchronous and asynchronous JavaScript execution. Provide examples of each.**

- **Synchronous Execution**: Code is executed sequentially, line by line. Each operation must finish before the next one starts. Synchronous code is blocking.
- *Example:* Simple variable assignment or arithmetic.
- **Asynchronous Execution**: Operations that don't block the main thread. These tasks (like fetching data, timers, file I/O) are offloaded, and JavaScript continues executing the rest of the code. A callback, Promise, or `async/await` is used to handle the result later.
- *Example:* `setTimeout()`, `fetch()` calls.

1. **What does DOM stand for in JavaScript, and why is it important?**
   DOM stands for **Document Object Model**.
   It is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM turns the web page into a tree structure of objects, allowing JavaScript to interact with and manipulate the HTML elements dynamically.

2. **Explain the concept of closures in JavaScript with a practical example.**
   A closure is the combination of a function and the lexical environment (the scope chain) in which that function was declared. This means that a function "remembers" and can access variables from its outer scope even after the outer function has finished executing.

- *Example:*

```
function makeCounter() {
let count = 0; // Outer variable
return function() {
count += 1; // Accessing the outer variable
return count;
};
}
const counter = makeCounter();
console.log(counter()); // Output: 1 (The inner function holds the scope of `count`)
```

React Theory
10. **Define the following core React concepts and explain their relationship: Components, Props, State.**
* **Components**: The fundamental building blocks of a React application. They are independent, reusable pieces of UI (User Interface) that return JSX (HTML-like structure).
* **Props (Properties)**: Used to pass data from a parent component down to a child component. Props are read-only and immutable within the child component.
* **State**: An object used to hold data that might change over time within a component. Changes in state trigger React to re-render the component. State is managed internally, typically using the `useState` hook.
* **Relationship**: Components are built using State (internal data management) and are configured and initialized

```
<!DOCTYPE html>
<html>
 <head>
  <header>Web app blah blah</header>
  <title>Web Lab</title>
  <style>
   header {
    padding-left: 20px;
    font-size: large;
    font-family: "Times New Roman", Times, serif;
   }
   body {
    align-items: center;
    margin: 0;
    font-family: Arial;
    background-color: aqua;
   }

   .box {
    margin: 20px;
   }
   .result {
    border: 1px solid black;
    padding: 10px;
    margin-top: 10px;
    width: 200px;
    background-color: rgb(215, 132, 193);
   }
  </style>
 </head>

 <body>
  <div class="box">
   <input id="inp" />
   <button onclick="run()">Submit</button>

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

```
    <script>
      function run() {
        let x = document.getElementById("inp").value;
        document.getElementById("out").innerText = x;
      }
    </script>
  </body>
</html>
```