LAB 2: Bootstrap, Javascript, Events and DOM Manipulation with Javascript

I. Bootstrap

Bootstrap is a popular front-end framework used for building responsive, mobile-first websites. It provides pre-designed components, such as navigation bars, forms, buttons, and grids, making it easier to develop web pages without starting from scratch. Bootstrap leverages HTML, CSS, and JavaScript, and helps developers create professional-looking layouts with minimal effort.

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>



    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
  </body>
</html>
```

1. Button Variants

Bootstrap provides different button styles using classes like btn-primary, btn-secondary, etc.

```html
<div>
  <button class="btn btn-primary">Primary</button>
  <button class="btn btn-secondary">Secondary</button>
  <button class="btn btn-success">Success</button>
  <button class="btn btn-danger">Danger</button>
  <button class="btn btn-warning">Warning</button>
  <button class="btn btn-info">Info</button>
  <button class="btn btn-light">Light</button>
  <button class="btn btn-dark">Dark</button>
</div>
```

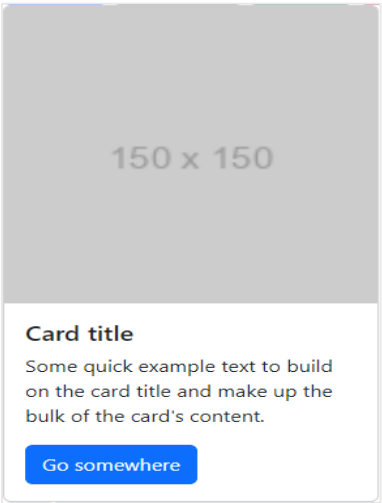
Result:

2. Card with Image and Text

Bootstrap's card component allows you to create content blocks with images, text, and actions.

```html
<div class="card" style="width: 18rem;">
  <img src="https://via.placeholder.com/150" class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some quick example text to build on the card title and make up the bulk of the card's content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Result:

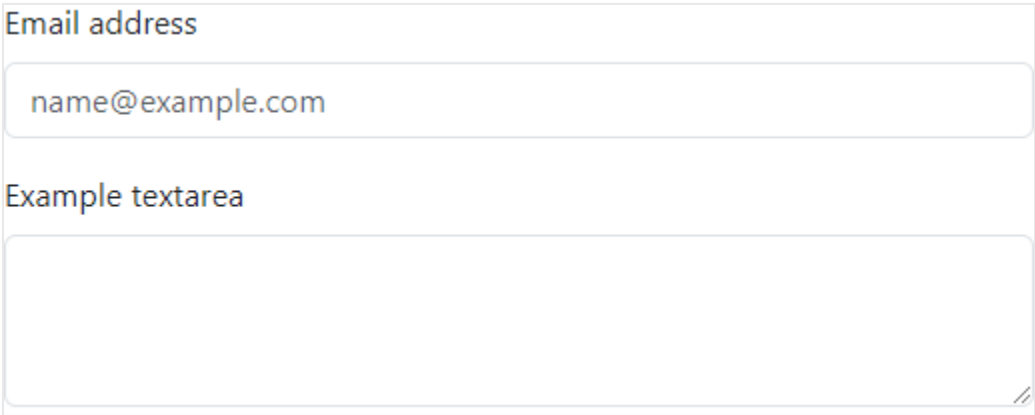


3. Form Input Group

Bootstrap's input group can be used to prepend or append buttons or text to an input field.

```html
<div class="mb-3">
  <label for="exampleFormControlInput1" class="form-label">Email address</label>
  <input type="email" class="form-control" id="exampleFormControlInput1" placeholder="name@example.com">
</div>
<div class="mb-3">
  <label for="exampleFormControlTextarea1" class="form-label">Example textarea</label>
  <textarea class="form-control" id="exampleFormControlTextarea1" rows="3"></textarea>
</div>
```

Result:

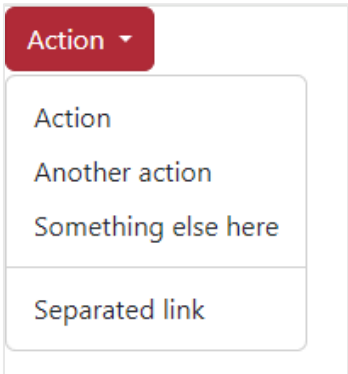


4. Navbar with Dropdown

A navigation bar with a dropdown menu is common in web development.

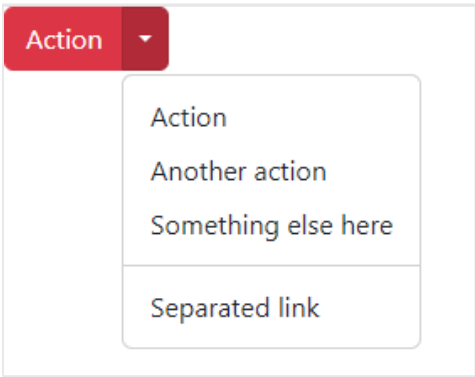
Single button:

```
<div class="btn-group">
  <button type="button" class="btn btn-danger dropdown-toggle" data-bs-toggle="dropdown" aria-expanded="false">
    Action
  </button>
  <ul class="dropdown-menu">
    <li><a class="dropdown-item" href="#">Action</a></li>
    <li><a class="dropdown-item" href="#">Another action</a></li>
    <li><a class="dropdown-item" href="#">Something else here</a></li>
    <li><hr class="dropdown-divider"></li>
    <li><a class="dropdown-item" href="#">Separated link</a></li>
  </ul>
</div>
```



Split button:

```
<div class="btn-group">
  <button type="button" class="btn btn-danger">Action</button>
  <button type="button" class="btn btn-danger dropdown-toggle dropdown-toggle-split" data-bs-toggle="dropdown" aria-expanded="false">
    <span class="visually-hidden">Toggle Dropdown</span>
  </button>
  <ul class="dropdown-menu">
    <li><a class="dropdown-item" href="#">Action</a></li>
    <li><a class="dropdown-item" href="#">Another action</a></li>
    <li><a class="dropdown-item" href="#">Something else here</a></li>
    <li><hr class="dropdown-divider"></li>
    <li><a class="dropdown-item" href="#">Separated link</a></li>
  </ul>
</div>
```



5. Alert Messages

Bootstrap's alert component allows you to create different types of alert messages.

```
<div class="alert alert-success" role="alert">
  This is a success alert—check it out!
</div>
<div class="alert alert-danger" role="alert">
  This is a danger alert—be cautious!
</div>
<div class="alert alert-warning" role="alert">
  This is a warning alert—watch out!
</div>
```

Result:

This is a success alert—check it out!

This is a danger alert—be cautious!

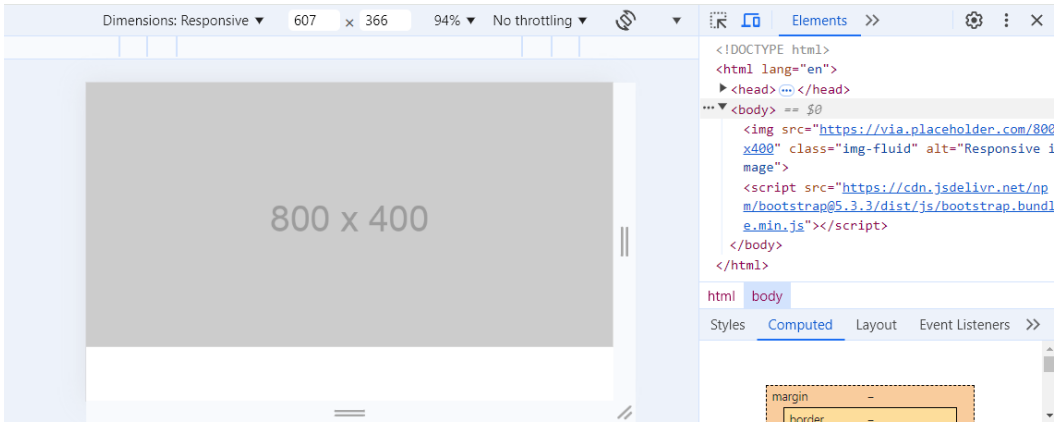
This is a warning alert—watch out!

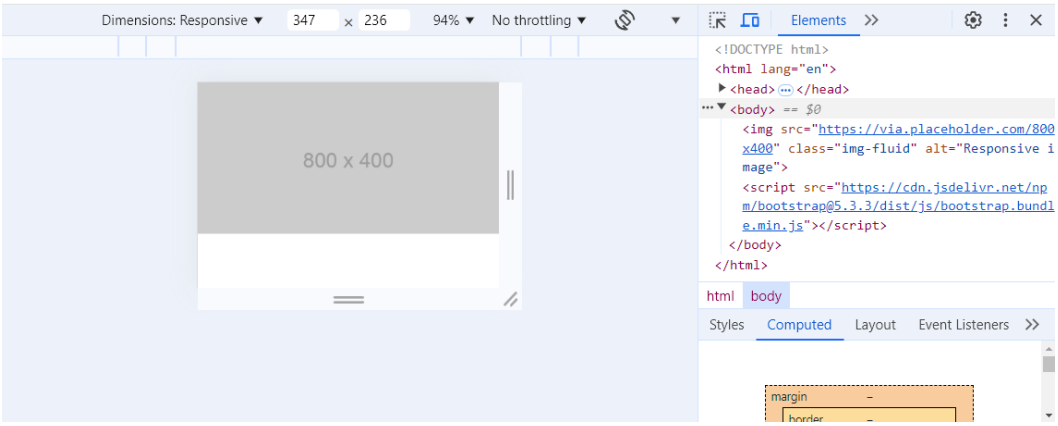
6. Responsive Image

Bootstrap's img-fluid class makes images responsive, adjusting based on screen size.

```
<img src="https://via.placeholder.com/800x400" class="img-fluid" alt="Responsive image">
```

Result:





7. Carousel for Images

The Bootstrap carousel component is used to create a slideshow.

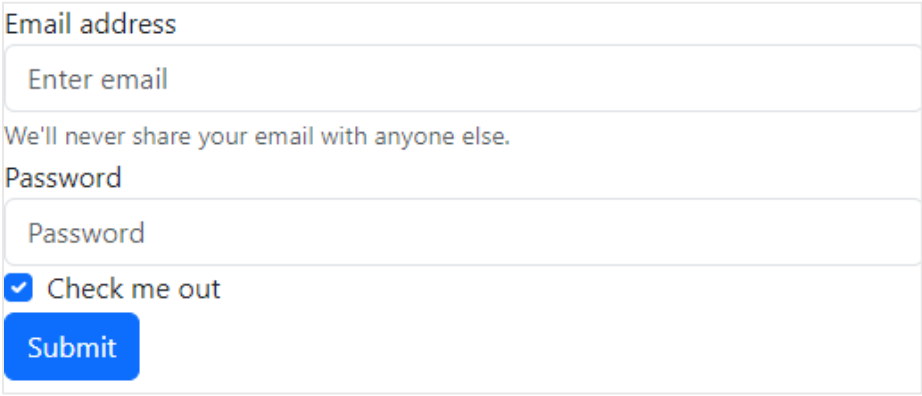
```html
<div id="carouselExample" class="carousel slide">
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="https://via.placeholder.com/800x400" class="d-block w-100" alt="Image 1">
    </div>
    <div class="carousel-item">
      <img src="https://via.placeholder.com/600x300" class="d-block w-100" alt="Image 2">
    </div>
    <div class="carousel-item">
      <img src="https://via.placeholder.com/500x300" class="d-block w-100" alt="Image 3">
    </div>
  </div>
  <button class="carousel-control-prev" type="button" data-bs-target="#carouselExample" data-bs-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button" data-bs-target="#carouselExample" data-bs-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Next</span>
  </button>
</div>
```

Result:

8. Form with Validation

Bootstrap can be used to create a styled form with validation feedback.

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" placeholder="Enter email">
    <small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone else.</small>
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1" placeholder="Password">
  </div>
  <div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Check me out</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

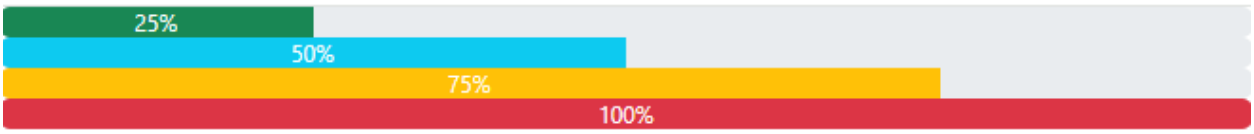


9. Progress Bar

Bootstrap's progress bar component shows the status of operations.

```
<div class="progress" role="progressbar" aria-label="Success example" aria-valuenow="25" aria-valuemin="0" aria-valuemax="100">
  <div class="progress-bar bg-success" style="width: 25%">25%</div>
</div>
<div class="progress" role="progressbar" aria-label="Info example" aria-valuenow="50" aria-valuemin="0" aria-valuemax="100">
  <div class="progress-bar bg-info" style="width: 50%">50%</div>
</div>
<div class="progress" role="progressbar" aria-label="Warning example" aria-valuenow="75" aria-valuemin="0" aria-valuemax="100">
  <div class="progress-bar bg-warning" style="width: 75%">75%</div>
</div>
<div class="progress" role="progressbar" aria-label="Danger example" aria-valuenow="100" aria-valuemin="0" aria-valuemax="100">
  <div class="progress-bar bg-danger" style="width: 100%">100%</div>
</div>
```

Result:

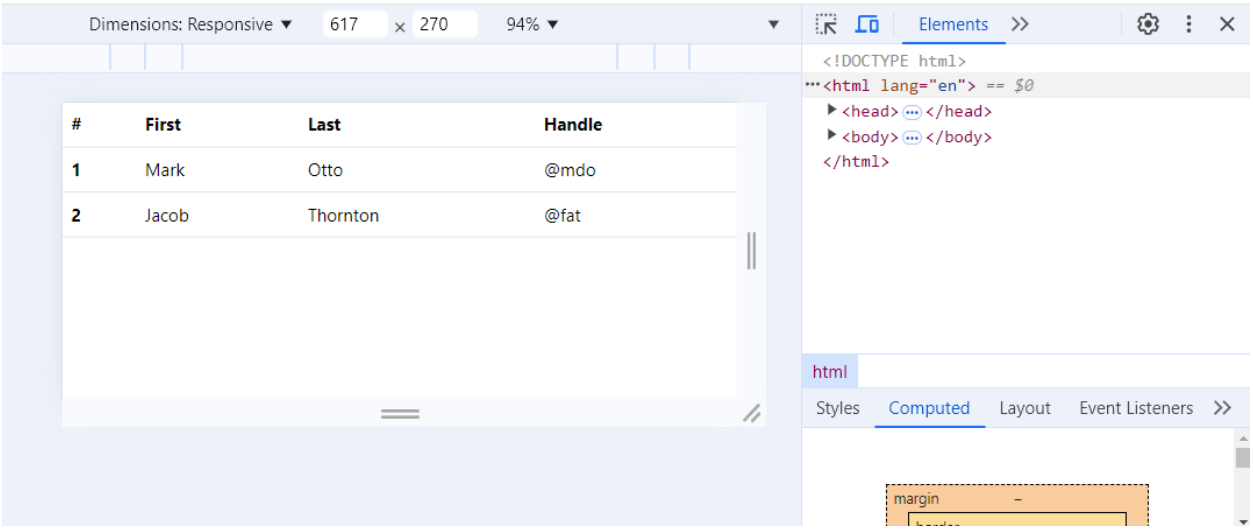


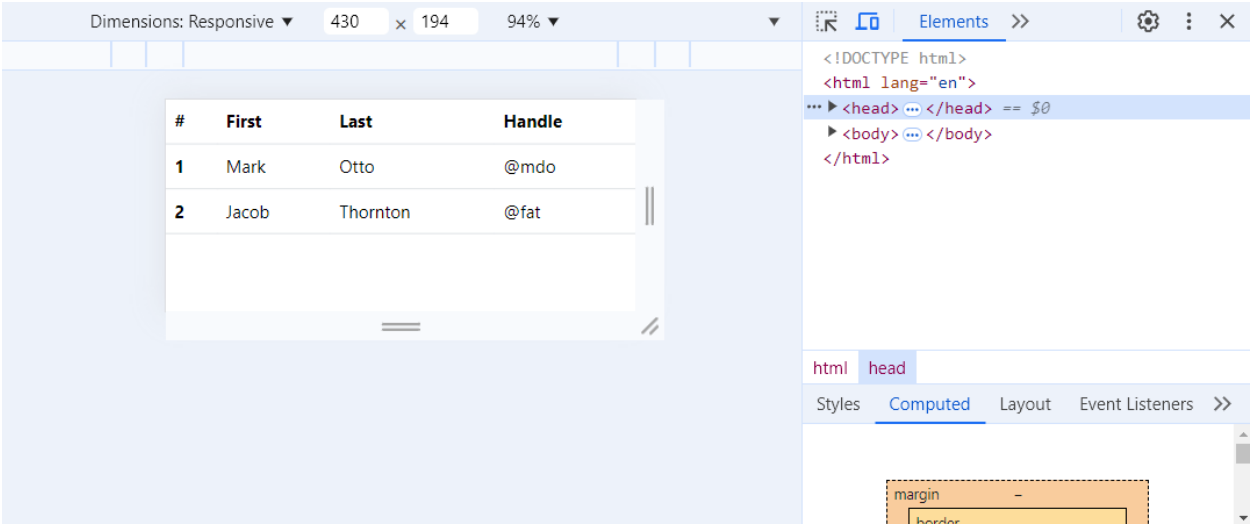
10. Responsive Table

Tables can be made responsive using Bootstrap's table-responsive class.

```
<div class="table-responsive">
  <table class="table">
    <thead>
      <tr>
        <th scope="col">#</th>
        <th scope="col">First</th>
        <th scope="col">Last</th>
        <th scope="col">Handle</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <th scope="row">1</th>
        <td>Mark</td>
        <td>Otto</td>
        <td>@mdo</td>
      </tr>
      <tr>
        <th scope="row">2</th>
        <td>Jacob</td>
        <td>Thornton</td>
        <td>@fat</td>
      </tr>
    </tbody>
  </table>
</div>
```

Result:

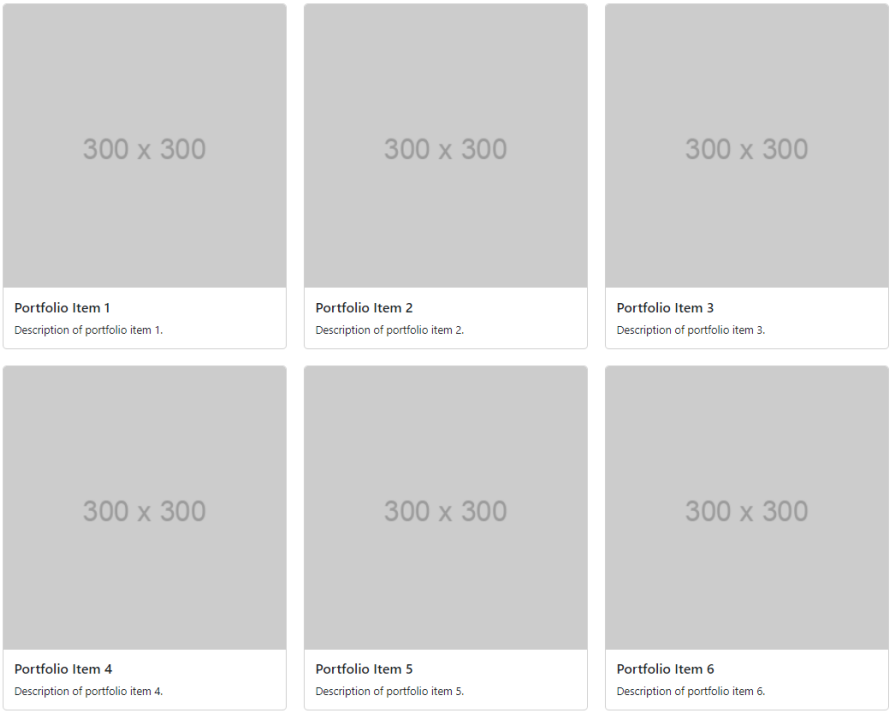


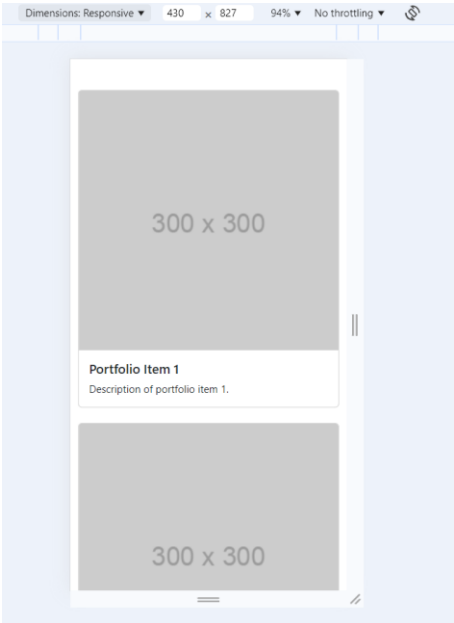


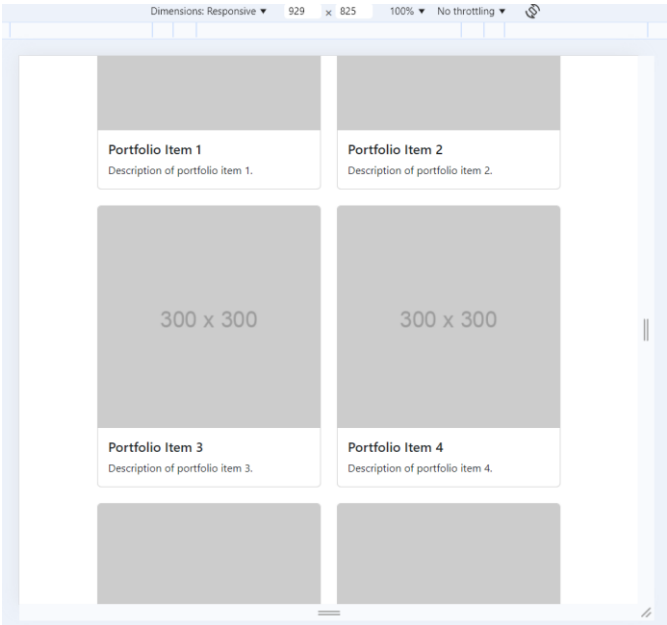
Exercise:

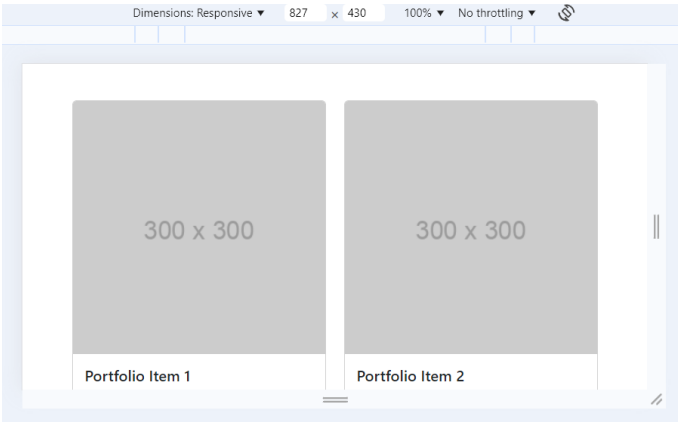
1. Create a Responsive Portfolio Grid

- **Objective**: Design a portfolio page using Bootstrap's grid system. The page should display 6 portfolio items arranged in 2 rows of 3 columns each on desktop. On smaller screens (tablets), it should display 2 items per row, and on mobile, only 1 item per row.
- **Concepts Covered**: Grid system, responsive design, container, row, and column classes.



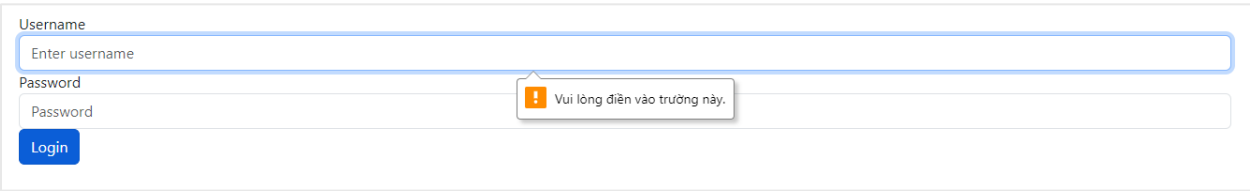






2. Create a Login Form with Validation Feedback

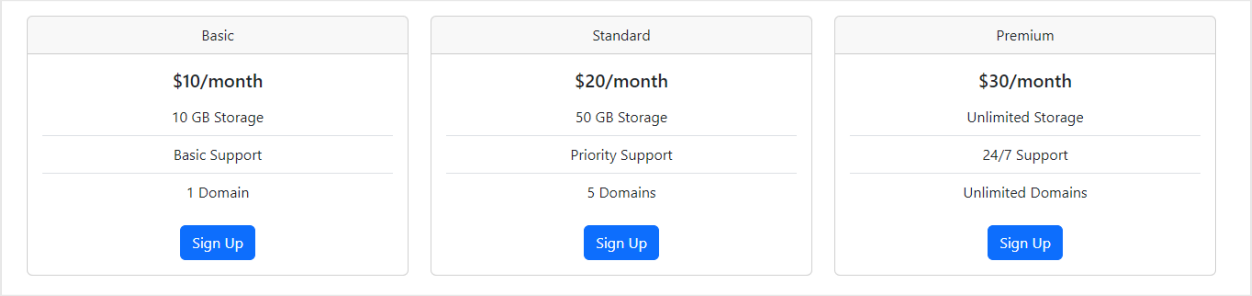
- **Objective**: Build a simple login form using Bootstrap components. The form should include fields for username and password. Add validation feedback for required fields (e.g., show a message when the user submits without filling in the fields).
- **Concepts Covered**: Forms, form validation, utility classes, buttons, form feedback.



3. Build a Pricing Table

- **Objective**: Create a simple pricing table using Bootstrap's grid system and card components. The table should display three different pricing options (e.g., Basic, Standard, Premium), each in its own card. Include a list of features, a price, and a "Sign Up" button for each card.

- **Concepts Covered**: Grid system, cards, buttons, typography, utility classes for spacing and alignment.



II. JavaScript

JavaScript is a high-level, interpreted scripting language primarily used to add interactivity to web pages.

1. Core Concepts

Variables and Data Types

- **Variables**: Declared using var, let, and const.
  - var has function scope, while let and const have block scope.
  - const is used for constants that should not be reassigned.
- **Data Types**:
  - Primitive: string, number, boolean, null, undefined, symbol.
  - Non-primitive: object, array, function.

```javascript
let age = 25;
const name = "Alice";
let isStudent = true;

console.log(age)
console.log(name)
console.log(isStudent)
```

```
25
Alice
true
```

Operators

- Arithmetic: +, -, *, /, %
- Comparison: ==, ===, !=, !==, <, >, <=, >=
- Logical: &&, ||, !

```
let a = 5, b = 10;
console.log(a + b); // 15
console.log(a === b); // false
console.log(a > b || a < b); // true
```

Control Structures

- **Conditionals**: if, else if, else, switch.
- **Loops**: for, while, do...while.

```
if (age > 18) {
    console.log("Adult");
  } else {
    console.log("Minor");
  }

  for (let i = 0; i < 5; i++) {
    console.log(i);
  }
```

Functions

- Can be declared using function declaration or function expression.
- Can also be defined as arrow functions.

```
function greet(name) {
    return `Hello, ${name}!`;
}

const sum = (a, b) => a + b;

console.log(greet("John")); // "Hello, John!"
console.log(sum(3, 7)); // 10
```

Arrays and Objects

- Arrays: List of values indexed by position.
- Objects: Collection of key-value pairs.

```
let fruits = ["apple", "banana", "cherry"];
let person = {
  name: "John",
  age: 30,
  city: "New York"
};

console.log(fruits[1]); // "banana"
console.log(person.name); // "John"
```

Example 1: Basic Calculator

- **Task**: Write a function that takes two numbers and an operator (+, -, *, /) and returns the result.

```
function calculate(num1, num2, operator) {
  switch (operator) {
    case '+':
      return num1 + num2;
    case '-':
      return num1 - num2;
    case '*':
      return num1 * num2;
    case '/':
      return num1 / num2;
    default:
      return "Invalid operator";
  }
}
```

Example 2: Array Filtering

- **Task**: Write a function that takes an array of numbers and returns a new array with only the even numbers.

```
function filterEvens(arr) {
  return arr.filter(num => num % 2 === 0);
}

console.log(filterEvens([1, 2, 3, 4, 5])); // [2, 4]
```

Example 3: Palindrome Checker

- **Task**: Write a function to check if a string is a palindrome (the same forwards and backwards).

```
function isPalindrome(str) {
  let reversed = str.split('').reverse().join('');
  return str === reversed;
}

console.log(isPalindrome("racecar")); // true
```

2. Events and DOM Manipulation

Events and DOM Manipulation are crucial concepts in JavaScript, as they allow you to make websites interactive by responding to user actions like clicks, form submissions, keyboard presses, and more. Let's break it down with more detail and provide practical examples.

The DOM (Document Object Model)

The DOM is a representation of an HTML document. It defines the structure of a web page as a tree of objects, where each HTML element is a node in that tree. JavaScript allows you to interact with and manipulate these nodes, changing the page dynamically without reloading.

Common DOM Methods

- **document.getElementById("id")**: Selects an element by its ID.
- **document.getElementsByClassName("class")**: Selects elements by their class name.
- **document.querySelector("selector")**: Selects the first element that matches a CSS selector.
- **document.querySelectorAll("selector")**: Selects all elements that match a CSS selector.
- **element.innerHTML**: Gets or sets the HTML content inside an element.
- **element.style**: Gets or sets the CSS styles of an element.

JavaScript Events

Events are actions that happen in the browser, such as a user clicking a button, pressing a key, submitting a form, or loading the page. JavaScript can "listen" for these events and execute code when they occur.

Common Event Types

- **Mouse Events**: click, dblclick, mouseover, mouseout, mousedown, mouseup
- **Keyboard Events**: keydown, keypress, keyup
- **Form Events**: submit, change, focus, blur
- **Window Events**: load, resize, scroll

Event Listeners

You can listen for events on elements by using the addEventListener method. This allows you to attach a function to an event on a specific element.

“element.addEventListener('event', function)”;

- **element**: The DOM element that you want to listen for events on.
- **'event'**: The type of event to listen for (e.g., 'click', 'keypress').
- **function**: The function that will run when the event is triggered.

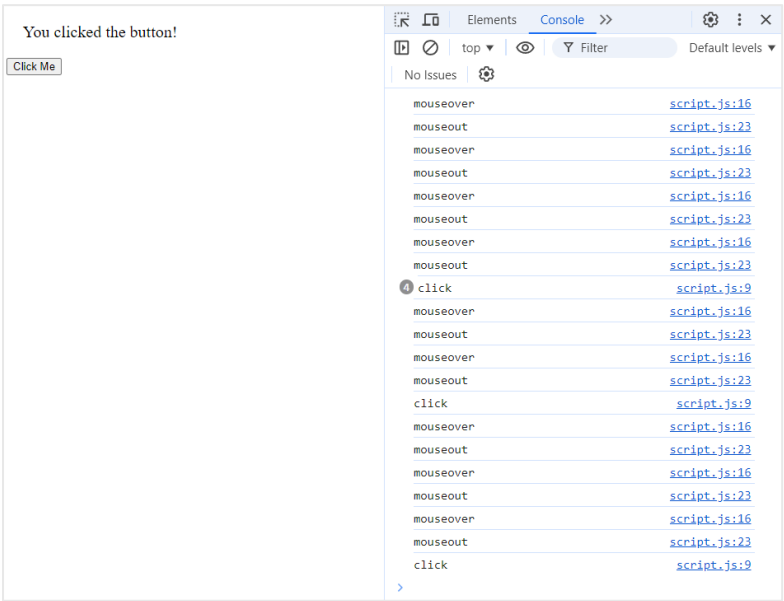
HTML Structure:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM Manipulation Example</title>
    <style>
        #myDiv {
            font-size: 20px;
            color: black;
            margin: 20px;
        }
    </style>
</head>
<body>
    <div id="myDiv">Original Text</div>
    <button id="myButton">Click Me</button>

    <script src="script.js"></script>
</body>
</html>
```

Javascript:

```javascript
// Get the DOM elements
const myDiv = document.getElementById('myDiv');
const myButton = document.getElementById('myButton');

// Add a 'click' event listener to the button
myButton.addEventListener('click', function() {
    // Change the content of the div
    myDiv.innerHTML = 'You clicked the button!';
    console.log('click')
});

// Add a 'mouseover' event listener to the div
myDiv.addEventListener('mouseover', function() {
    // Change the text color when hovering over the div
    myDiv.style.color = 'blue';
    console.log('mouseover')
});

// Add a 'mouseout' event listener to reset the color when mouse leaves
myDiv.addEventListener('mouseout', function() {
    // Reset the text color
    myDiv.style.color = 'black';
    console.log('mouseout')
});
```

Result:



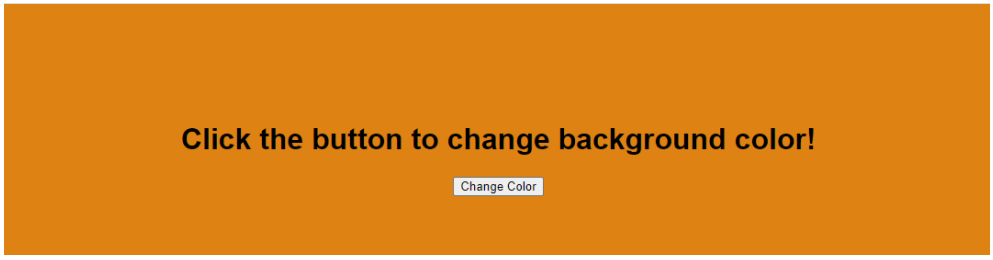
Example:

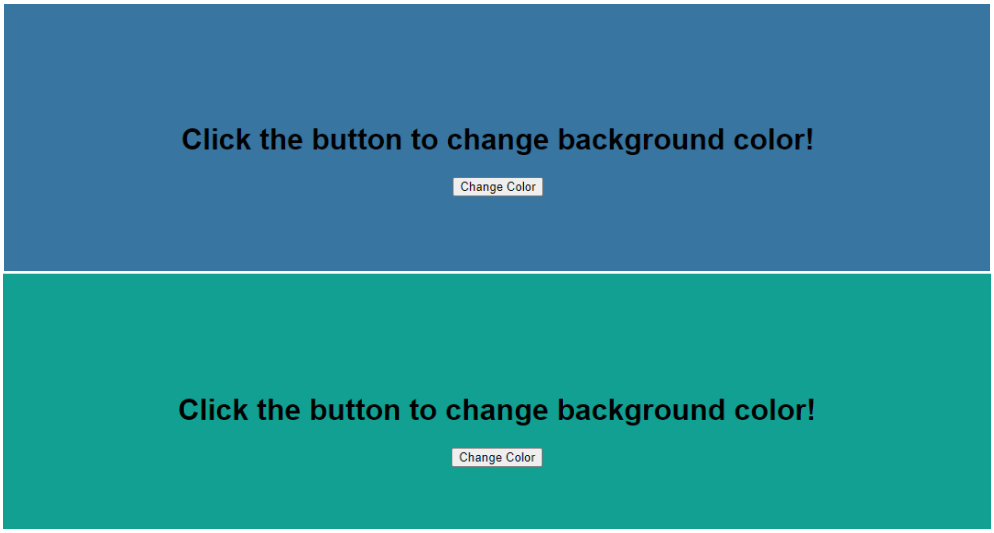
1. Change Background Color with Button Click

- **Objective**: Create a webpage with a button. When the button is clicked, the background color of the webpage should change to a random color.
- **Concepts Covered**: DOM manipulation, event handling, working with CSS styles.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Change Background Color</title>
  <style>
    body {
      text-align: center;
      padding: 100px;
      font-family: Arial, sans-serif;
    }
  </style>
</head>
<body>

  <h1>Click the button to change background color!</h1>
  <button id="colorButton">Change Color</button>

  <script>
    const button = document.getElementById('colorButton');
    button.addEventListener('click', function() {
      const randomColor = '#' + Math.floor(Math.random() * 16777215).toString(16);
      document.body.style.backgroundColor = randomColor;
    });
  </script>

</body>
</html>
```

Result:





2. Simple Counter

- **Objective**: Create a simple counter with "Increase" and "Decrease" buttons to increment or decrement a number displayed on the webpage.
- **Concepts Covered**: DOM manipulation, event listeners, working with variables.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Counter</title>
  <style>
    body {
      text-align: center;
      padding: 100px;
      font-family: Arial, sans-serif;
    }
    #counter {
      font-size: 48px;
      margin: 20px;
    }
  </style>
</head>
<body>
  <h1>Simple Counter</h1>
  <div id="counter">0</div>
  <button id="increase">Increase</button>
  <button id="decrease">Decrease</button>
  <script>
    let count = 0;
    const counter = document.getElementById('counter');
    document.getElementById('increase').addEventListener('click', () => {
      count++;
      counter.innerText = count;
    });
    document.getElementById('decrease').addEventListener('click', () => {
      count--;
      counter.innerText = count;
    });
  </script>

</body>
</html>
```

3. Form Validation

- **Objective**: Create a form with fields for "Name" and "Email". When the form is submitted, display an error message if any field is left empty.
- **Concepts Covered**: Form validation, event handling, DOM manipulation.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form Validation</title>
  <style>
    body {
      text-align: center;
      padding: 100px;
      font-family: Arial, sans-serif;
    }
    #error {
      color: red;
      display: none;
    }
  </style>
</head>
```

```html
<body>
  <h1>Form Validation</h1>
  <form id="myForm">
    <div>
      <label for="name">Name:</label>
      <input type="text" id="name" name="name">
    </div>
    <div>
      <label for="email">Email:</label>
      <input type="email" id="email" name="email">
    </div>
    <button type="submit">Submit</button>
  </form>
  <p id="error">Please fill in all fields!</p>

  <script>
    const form = document.getElementById('myForm');
    const error = document.getElementById('error');

    form.addEventListener('submit', function(e) {
      e.preventDefault();
      const name = document.getElementById('name').value;
      const email = document.getElementById('email').value;

      if (!name || !email) {
        error.style.display = 'block';
      } else {
        error.style.display = 'none';
        alert('Form submitted successfully!');
      }
    });
  </script>

</body>
</html>
```

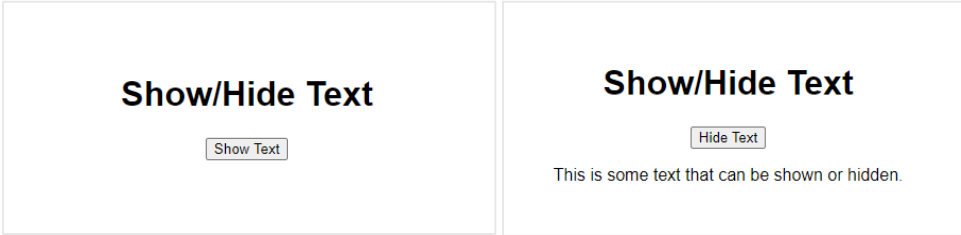
Result:

4. Show/Hide Text with Button

- **Objective**: Create a webpage with some text and a button to toggle between showing and hiding the text.
- **Concepts Covered**: DOM manipulation, event handling.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Show/Hide Text</title>
  <style>
    body {
      text-align: center;
      padding: 100px;
      font-family: Arial, sans-serif;
    }
  </style>
</head>
<body>

  <h1>Show/Hide Text</h1>
  <button id="toggleButton">Hide Text</button>
  <p id="text">This is some text that can be shown or hidden.</p>

  <script>
    const button = document.getElementById('toggleButton');
    const text = document.getElementById('text');

    button.addEventListener('click', function() {
      if (text.style.display === 'none') {
        text.style.display = 'block';
        button.innerText = 'Hide Text';
      } else {
        text.style.display = 'none';
        button.innerText = 'Show Text';
      }
    });
  </script>

</body>
</html>
```

Result:

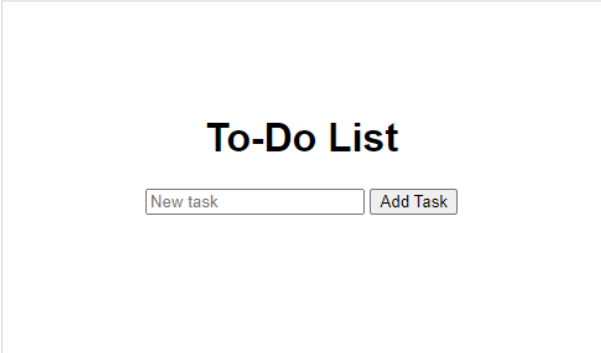


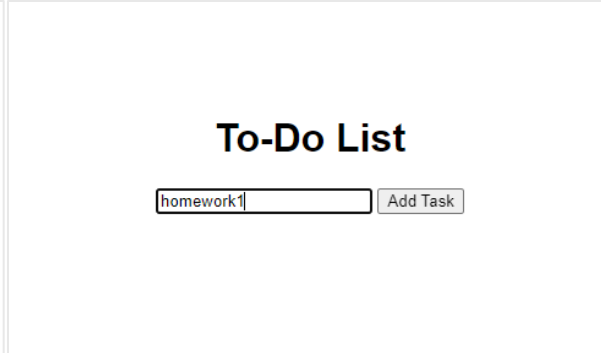
5. Simple To-Do List

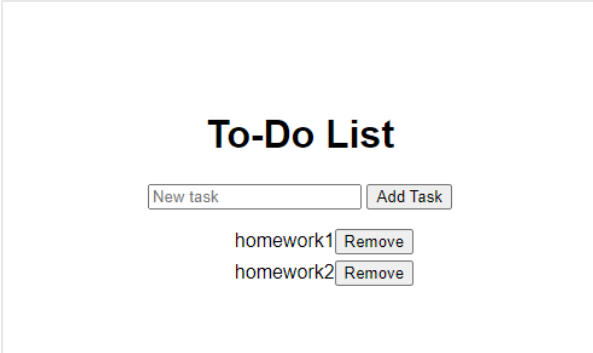
- **Objective**: Create a simple to-do list where the user can add new tasks and remove completed tasks.
- **Concepts Covered**: DOM manipulation, event listeners, working with arrays.

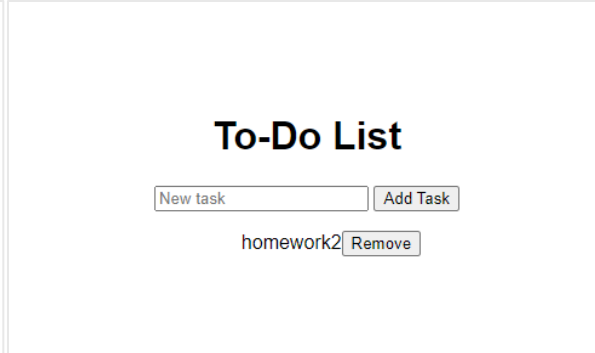
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>To-Do List</title>
  <style>
    body {
      text-align: center;
      padding: 100px;
      font-family: Arial, sans-serif;
    }
    ul {
      list-style-type: none;
    }
    li {
      margin: 5px 0;
    }
  </style>
</head>
```

```html
<body>

  <h1>To-Do List</h1>
  <input type="text" id="taskInput" placeholder="New task">
  <button id="addTask">Add Task</button>

  <ul id="taskList"></ul>

  <script>
    const addTaskButton = document.getElementById('addTask');
    const taskList = document.getElementById('taskList');

    addTaskButton.addEventListener('click', function() {
      const taskInput = document.getElementById('taskInput').value;
      if (taskInput) {
        const li = document.createElement('li');
        li.textContent = taskInput;

        // Add button to remove task
        const removeButton = document.createElement('button');
        removeButton.textContent = 'Remove';
        removeButton.addEventListener('click', function() {
          taskList.removeChild(li);
        });

        li.appendChild(removeButton);
        taskList.appendChild(li);
        document.getElementById('taskInput').value = ''; // clear input
      }
    });
  </script>

</body>
```







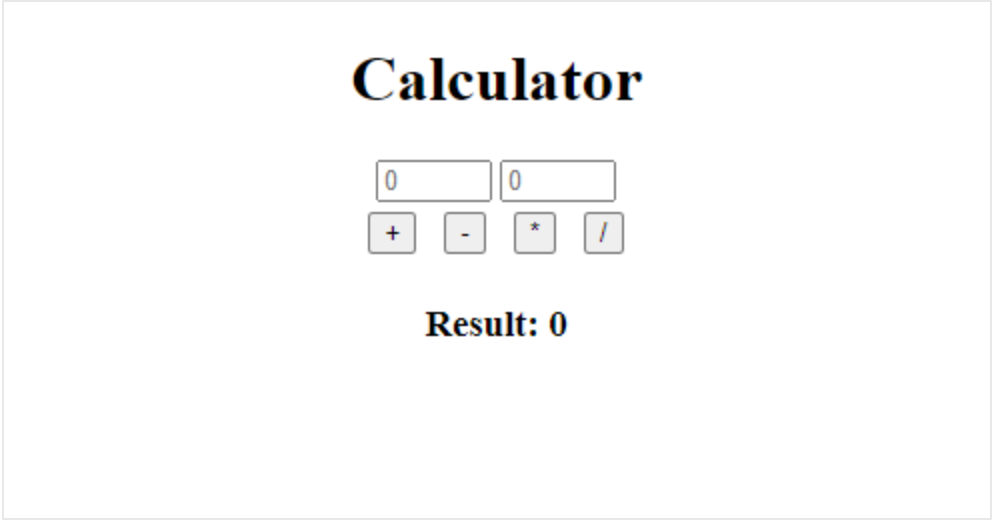


Exercise:

1. Calculator Application

- **Objective**: Create a basic calculator that can perform addition, subtraction, multiplication, and division.
- **Concepts Covered**: Event handling, DOM manipulation, functions.

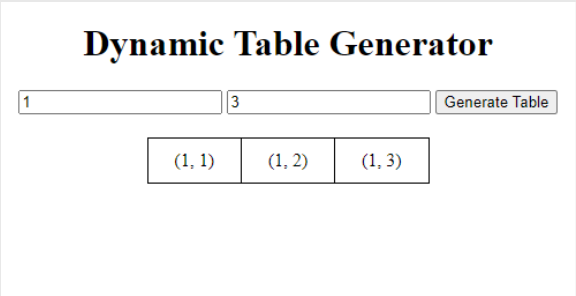
```html
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Calculator</title>
  <style>
    .calculator {
      max-width: 300px;
      margin: 0 auto;
      text-align: center;
    }
    input {
      width: 50px;
    }
    button {
      margin: 5px;
    }
  </style>
</head>
<body>

  <div class="calculator">
    <h1>Calculator</h1>
    <input type="number" id="num1" placeholder="0">
    <input type="number" id="num2" placeholder="0">
    <br>
    <button id="add">+</button>
    <button id="subtract">-</button>
    <button id="multiply">*</button>
    <button id="divide">/</button>
    <h3>Result: <span id="result">0</span></h3>
  </div>
```

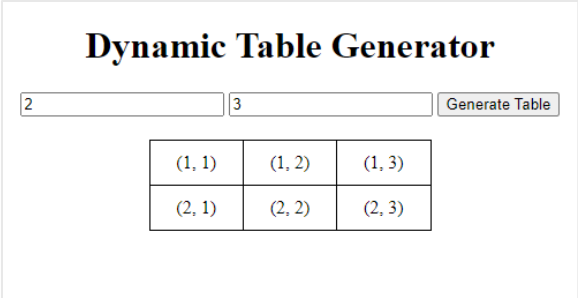


2. Dynamic Table Generator

- **Objective**: Create a webpage where the user can input the number of rows and columns for a table. Upon clicking a button, a table with the specified dimensions is dynamically created.
- **Concepts Covered**: DOM manipulation, loops, event handling.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dynamic Table</title>
  <style>
    table {
      width: 50%;
      margin: 20px auto;
      border-collapse: collapse;
    }
    table, th, td {
      border: 1px solid black;
    }
    td {
      padding: 10px;
      text-align: center;
    }
  </style>
</head>
<body>

  <div style="text-align:center;">
    <h1>Dynamic Table Generator</h1>
    <input type="number" id="rows" placeholder="Rows">
    <input type="number" id="cols" placeholder="Columns">
    <button id="generate">Generate Table</button>
  </div>

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

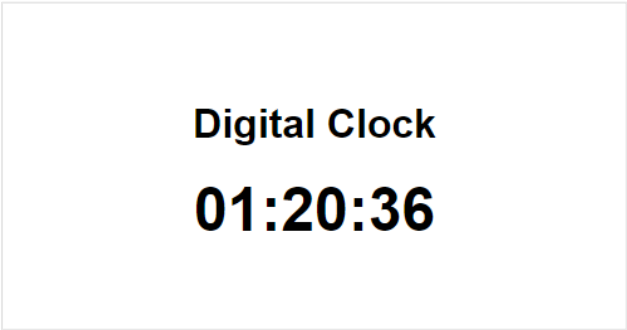




3. Digital Clock

- **Objective**: Create a digital clock that updates every second and displays the current time in HH:MM:SS format.
- **Concepts Covered**: setInterval, working with Date objects, DOM manipulation.

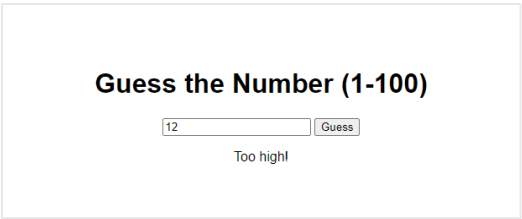
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Digital Clock</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      padding: 50px;
    }
    #clock {
      font-size: 48px;
      font-weight: bold;
    }
  </style>
</head>
<body>

  <h1>Digital Clock</h1>
  <div id="clock"></div>
```

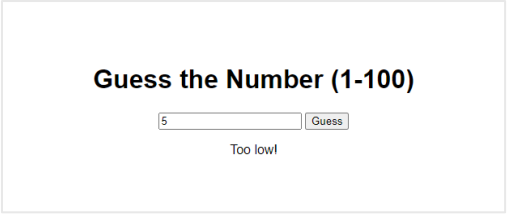


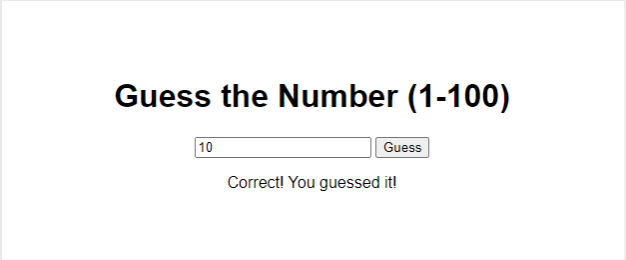
4. Guessing Game

- **Objective**: Create a number guessing game. The computer selects a random number between 1 and 100, and the user tries to guess it. The user is given feedback after each guess (too high, too low, or correct).
- **Concepts Covered**: Math.random(), conditionals, event handling, DOM manipulation.

```html
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
  <title>Number Guessing Game</title>
  <style>
    body {
      text-align: center;
      padding: 50px;
      font-family: Arial, sans-serif;
    }
  </style>
</head>
<body>

  <h1>Guess the Number (1-100)</h1>
  <input type="number" id="guess" placeholder="Enter your guess">
  <button id="submitGuess">Guess</button>
  <p id="feedback"></p>
```



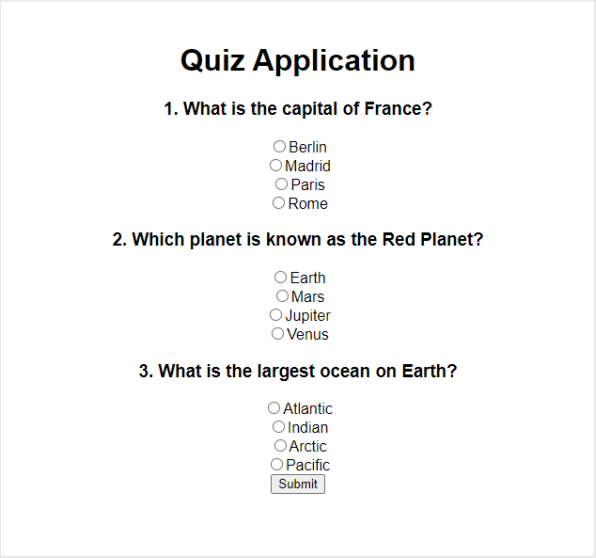


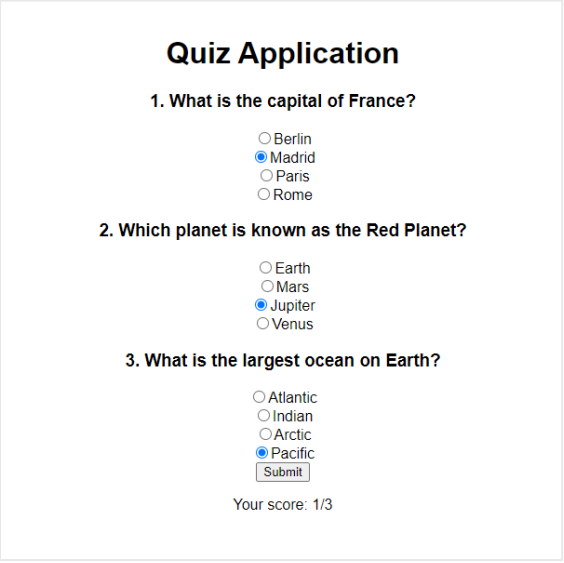


5. Quiz Application with Multiple Questions

- **Objective**: Create a simple quiz application that displays multiple questions with multiple-choice answers. After the user submits, it shows the score.
- **Concepts Covered**: Arrays, DOM manipulation, event handling.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Quiz Application</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      padding: 20px;
    }
  </style>
</head>
<body>

  <h1>Quiz Application</h1>
  <div id="quiz"></div>
  <button id="submit">Submit</button>
  <p id="result"></p>
```





<u>HOMEWORK:</u>

1. Complete all exercises in the Exercise section

2. OOP exercise:

You will learn about OOP in JavaScript on your own and complete the exercise below

Library System Simulation

Task Overview:

You are tasked with building a library system that:

1. Manages books and users.
2. Allows users to borrow and return books.
3. Enforces different borrowing rules based on the type of user (e.g., student vs teacher).
4. Tracks the borrowed status of books and manages a user's borrowing limits.

Requirements:

1. **Book Class**:
    - Each book should have:
        - **title (string)**
        - **author (string)**
        - **isbn (string)**
        - **availableCopies (integer)** — the number of available copies.
    - Methods:
        - **borrowBook()** — Decreases available copies by 1.
        - **returnBook()** — Increases available copies by 1.
2. **User Class (Abstract)**:
    - Each user should have:
        - **name (string)**
        - **userType (string)** — Should be either "Student" or "Teacher".
        - **borrowedBooks (array)** — List of books the user has borrowed.
    - Abstract Method:
        - **borrow(book)** — Should enforce user-specific borrowing rules.
        - **return(book)** — Should return a borrowed book.
3. **Student Class (inherits from User)**:
    - A student can borrow up to 3 books at a time.
    - Implements **borrow()** method to check borrowing limit.
4. **Teacher Class (inherits from User)**:
    - A teacher can borrow up to 5 books at a time.
    - Implements **borrow()** method to check borrowing limit.
5. **Library Class**:
    - Manages a collection of books and users.

- Methods:
    - **addBook(book)** — Adds a book to the library.
    - **addUser(user)** — Adds a user to the system.
    - **borrowBook(user, book)** — Handles the borrowing process.
    - **returnBook(user, book)** — Handles the returning process.
    - **listAvailableBooks()** — Lists all available books.

Additional Rules:

- A user cannot borrow a book if there are no available copies.
- A user cannot borrow more than their limit of books.
- The library should throw meaningful errors (exceptions) when an action is not allowed, such as exceeding borrow limits or borrowing an unavailable book.