ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

☑ INTRODUCTION TO JAVASCRIPT – ADDING INTERACTIVITY TO WEBSITES

CHAPTER 1: WHAT IS JAVASCRIPT?

1.1 Understanding JavaScript

JavaScript (**JS**) is a **programming language** used to make websites **interactive**. Unlike HTML (which structures the page) and CSS (which styles the page), JavaScript adds **dynamic behaviors** like:

✓ Animations

✓ User interactions (clicks, form submissions)

✓ Real-time updates (fetching data without reloading the page)

📌 **Example:** Changing text when a button is clicked.

```
<button onclick="document.getElementById('demo').innerHTML = 'Hello, JavaScript!'">
  Click Me
</button>

<p id="demo"></p>
```

✓ **Effect:** Clicking the button updates the paragraph content dynamically.

CHAPTER 2: HOW JAVASCRIPT WORKS IN WEBSITES

JavaScript runs in the **browser** alongside HTML and CSS. It can be:

- **Inline (inside an HTML tag)**
- **Internal (within a <script> tag in HTML)**
- **External (linked as a separate file)**

2.1 Adding JavaScript to an HTML Page

1. Inline JavaScript (Inside an HTML tag)

```
<button onclick="alert('Button Clicked!')">Click Me</button>
```

✓ **Effect:** Clicking the button triggers an **alert message**.

2. Internal JavaScript (Inside <script> in HTML)

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Example</title>
</head>
<body>
  <p id="message">Click the button to change this text.</p>
  <button onclick="changeText()">Click Me</button>

  <script>
```

```
    function changeText() {
      document.getElementById("message").innerHTML = "Text changed!";
    }
  </script>
</body>
</html>
```

✓ **Effect:** Clicking the button updates the paragraph text.

3. External JavaScript (Separate .js file)

index.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>External JavaScript</title>
  <script src="script.js"></script>  <!-- Linking JavaScript file -->
</head>
<body>
  <h1 id="title">Hello World</h1>
  <button onclick="changeTitle()">Change Title</button>
</body>
</html>
```

script.js (External File):

```
function changeTitle() {
   document.getElementById("title").innerHTML = "JavaScript is Awesome!";
}
```

✓ **Effect:** Clicking the button changes the heading dynamically.

CHAPTER 3: JAVASCRIPT BASICS

3.1 Variables in JavaScript

Variables store data values.

📌 **Declaring Variables:**

```
let name = "Alice"; // String
const age = 25;  // Number
var isStudent = true; // Boolean
```

✓ **Use let or const instead of var for better scope control.**

3.2 Data Types in JavaScript

JavaScript supports various **data types,** including:

Data Type	Example
String	"Hello, World!"
Number	42, 3.14

Boolean	true, false
Array	["Apple", "Banana", "Cherry"]
Object	{name: "John", age: 30}

📌 **Example: Using Different Data Types**

```
let person = { name: "Alice", age: 25, city: "New York" };

console.log(person.name); // Output: Alice
```

3.3 Functions in JavaScript

Functions are reusable blocks of code that **perform a specific task**.

📌 **Example: Function to Add Two Numbers**

```
function addNumbers(a, b) {
   return a + b;
}

console.log(addNumbers(5, 3));  // Output: 8
```

✓ **Functions help avoid repeating code and improve readability.**

CHAPTER 4: DOM MANIPULATION – CHANGING WEBSITE CONTENT DYNAMICALLY

The **DOM (Document Object Model)** allows JavaScript to change **HTML elements dynamically**.

4.1 Changing Text Content

📌 **Example:**

```
document.getElementById("demo").innerHTML = "Text Changed!";
```

4.2 Changing Styles

📌 **Example:**

```
document.getElementById("demo").style.color = "blue";
```

4.3 Hiding & Showing Elements

📌 **Example:**

```
document.getElementById("demo").style.display = "none"; // Hides the element

document.getElementById("demo").style.display = "block"; // Shows the element
```

✓ **Practical Use:** Interactive UI features like **showing notifications** or **hiding elements**.

CHAPTER 5: JAVASCRIPT EVENTS – ADDING INTERACTIVITY

Events allow JavaScript to respond to user actions.

Event	Triggered When
onclick	User clicks an element
onmouseover	Mouse hovers over an element
onmouseout	Mouse leaves an element
onkeyup	User releases a key

📌 **Example: Change Color on Button Click**

```
<button onclick="changeColor()">Click Me</button>


<script>
function changeColor() {
  document.body.style.backgroundColor = "lightblue";
}
</script>
```

✓ **Effect:** Clicking the button changes the **background color**.

CHAPTER 6: JAVASCRIPT FORM VALIDATION

JavaScript can **validate forms** before submission.

📌 **Example: Check if Name is Entered**

```
<form onsubmit="return validateForm()">
  <input type="text" id="name" placeholder="Enter your name">
  <button type="submit">Submit</button>
</form>


<script>
function validateForm() {
  let name = document.getElementById("name").value;
  if (name === "") {
```

```
    alert("Name is required!");
    return false;
  }
  return true;
}
</script>
```

✓ **Effect:** Shows an alert if the input field is empty.

CHAPTER 7: EXERCISE

7.1 Multiple Choice Questions

1. What is the correct way to declare a variable in JavaScript?
   - (a) variable name = "John";
   - (b) var name = "John";
   - (c) let name = "John";
   - (d) Both (b) and (c) ✅
2. Which JavaScript function is used to display a message?
   - (a) alert() ✅
   - (b) message()
   - (c) console.log()
   - (d) popup()
3. What is the purpose of document.getElementById("demo")?

- (a) Select an element by tag name
- (b) Select an element by class
- (c) Select an element by ID ✅
- (d) Create a new element

7.2 Practical Tasks

📌 **Task 1: Create a webpage where clicking a button changes the text color.**
📌 **Task 2: Write a JavaScript function to validate a form that requires a name and email.**
📌 **Task 3: Use JavaScript to hide and show an element when clicking a button.**

CHAPTER 8: SUMMARY

✅ **JavaScript adds interactivity** to web pages.
✅ It can **change HTML content dynamically** using the **DOM**.
✅ **Events** allow JavaScript to respond to **user actions**.
✅ **Functions** help organize and reuse code.
✅ JavaScript can **validate form input** before submission.

☑ VARIABLES, DATA TYPES & FUNCTIONS IN JAVASCRIPT

CHAPTER 1: INTRODUCTION TO JAVASCRIPT VARIABLES, DATA TYPES, AND FUNCTIONS

JavaScript is a powerful programming language that enables developers to create **dynamic and interactive** web applications.

- **Variables** store data that can be used and manipulated in a program.
- **Data types** define the kind of data stored in a variable.
- **Functions** are reusable blocks of code that perform specific tasks.

This chapter will explore these fundamental concepts in detail.

CHAPTER 2: VARIABLES IN JAVASCRIPT

2.1 What is a Variable?

A **variable** is a container that stores data. In JavaScript, variables allow us to store numbers, text, objects, and more.

2.2 Declaring Variables in JavaScript

There are three ways to declare variables in JavaScript:
✓ var – Global or function-scoped (older method).
✓ let – Block-scoped and preferred for modern code.
✓ const – Block-scoped and cannot be reassigned.

📌 **Example:**

var name = "Alice"; // Can be redeclared and updated (not recommended)

let age = 25; // Can be updated but not redeclared

const country = "USA"; // Cannot be changed

2.3 Rules for Naming Variables

1. Must **start with a letter, _ (underscore), or $ (dollar sign)**.
2. Cannot start with a **number** (e.g., 2name is invalid).
3. Case-sensitive (name and Name are different variables).
4. Cannot use **reserved keywords** (e.g., let let = 10; is invalid).

✓ Good Variable Names:

let userName = "John";

let _score = 50;

let $price = 99.99;

✓ Bad Variable Names:

let 2name = "Error"; // ❌ Starts with a number

let let = 10; // ❌ Uses a reserved keyword

CHAPTER 3: DATA TYPES IN JAVASCRIPT

3.1 What are Data Types?

Data types define the kind of value a variable holds. JavaScript has **two categories** of data types:

1. **Primitive Data Types** (Stores a single value):
    - String – Text data ("Hello", 'JavaScript').
    - Number – Integers & decimals (42, 3.14).
    - Boolean – True or False (true, false).
    - Undefined – Variable declared but not assigned (let x;).
    - Null – Represents an empty value (let y = null;).
2. **Non-Primitive Data Types** (Stores collections of data):
    - Array – Stores multiple values (["Apple", "Banana", "Cherry"]).
    - Object – Stores key-value pairs ({name: "John", age: 30}).

3.2 Examples of Data Types in JavaScript

📌 **Primitive Data Types:**

```
let name = "Alice";  // String

let age = 25;  // Number

let isStudent = false;  // Boolean

let score;  // Undefined

let emptyValue = null;  // Null
```

📌 **Non-Primitive Data Types:**

```
// Array
```

```
let fruits = ["Apple", "Banana", "Cherry"];


// Object
let person = { name: "John", age: 30, city: "New York" };
```

3.3 Checking Data Types

Use the typeof operator to check the data type of a variable.

📌 **Example:**

```
console.log(typeof "Hello");  // Output: string
console.log(typeof 42);  // Output: number
console.log(typeof true);  // Output: boolean
console.log(typeof undefined);  // Output: undefined
console.log(typeof null);  // Output: object (special case)
console.log(typeof {});  // Output: object
console.log(typeof []);  // Output: object (Arrays are objects)
```

CHAPTER 4: FUNCTIONS IN JAVASCRIPT

4.1 What is a Function?

A **function** is a reusable block of code that performs a specific task.

- Functions make code **modular and efficient**.
- Functions can **accept inputs** (parameters) and **return outputs**.

4.2 Defining a Function in JavaScript

There are multiple ways to define functions in JavaScript:

1. Function Declaration

📌 **Example:**

```
function greet() {
   console.log("Hello, welcome to JavaScript!");
}
greet();  // Calling the function
```

✓ **Effect:** Prints "Hello, welcome to JavaScript!" to the console.

---

2. Function with Parameters

📌 **Example:**

```
function add(a, b) {
   return a + b;
}
console.log(add(5, 3));  // Output: 8
```

✓ **Effect:** Returns and prints the sum of 5 + 3.

---

3. Function Expressions (Assigning a function to a variable)

📌 **Example:**

```
const multiply = function(x, y) {
  return x * y;
};
console.log(multiply(4, 2));  // Output: 8
```

✓ **Effect:** The function is stored in a variable and called using multiply().

---

4. Arrow Functions (Modern ES6 Syntax)

📌 **Example:**

```
const square = (num) => num * num;
console.log(square(6));  // Output: 36
```

✓ **Effect:** Returns the square of 6.

4.3 Default Parameters in Functions

📌 **Example:**

```
function greet(name = "Guest") {
  console.log("Hello, " + name);
}
greet();  // Output: Hello, Guest
greet("Alice");  // Output: Hello, Alice
```

✓ **Effect:** If no argument is passed, "Guest" is used as a default value.

CHAPTER 5: EXERCISE

5.1 Multiple Choice Questions

1. Which keyword is used to declare a variable that cannot be changed?
    - (a) let
    - (b) var
    - (c) const ✅
    - (d) function
2. What will be the output of typeof null?
    - (a) null
    - (b) object ✅
    - (c) undefined
    - (d) string
3. How do you call a function named sayHello?
    - (a) sayHello;
    - (b) sayHello(); ✅
    - (c) call sayHello();
    - (d) execute sayHello();

5.2 Practical Tasks

📌 **Task 1: Declare a variable userAge, assign it a number, and print its type.**

📌 **Task 2: Write a function calculateArea that takes width and height as parameters and returns the area.**

📌 **Task 3: Create an arrow function doubleNumber that takes a number and returns its double.**

CHAPTER 6: SUMMARY

✅ **Variables** store data and are declared using var, let, or const.
✅ **Data Types** include **strings, numbers, booleans, arrays, and objects**.
✅ **Functions** perform specific tasks and can take **parameters** and **return values**.
✅ **Arrow functions** provide a shorter syntax for defining functions.
✅ JavaScript helps make webpages **dynamic and interactive**.

☑ EVENT HANDLING & DOM MANIPULATION IN JAVASCRIPT

CHAPTER 1: INTRODUCTION TO EVENT HANDLING & DOM MANIPULATION

1.1 What is the DOM?

The **Document Object Model (DOM)** is a programming interface that represents the structure of a webpage. It allows JavaScript to:

✓ Access and modify **HTML elements** dynamically.

✓ Change **styles, text, and attributes** of elements.

✓ Handle **user interactions** like clicks, input, or keypresses.

📌 **Example of a Simple HTML Structure and its DOM Representation:**

```
<!DOCTYPE html>
<html>
<head>
  <title>My Page</title>
</head>
<body>
  <h1 id="title">Hello, World!</h1>
</body>
</html>
```

✓ The browser creates a **DOM Tree** where <h1> is a **node** that can be accessed and modified with JavaScript.

CHAPTER 2: SELECTING ELEMENTS IN THE DOM

To manipulate elements in the DOM, we must first **select** them. JavaScript provides multiple methods for selecting elements:

2.1 getElementById() – Selects an Element by ID

📌 **Example:**

```
let title = document.getElementById("title");

console.log(title.innerHTML);  // Output: Hello, World!
```

✓ This selects the <h1> element with id="title".

2.2 getElementsByClassName() – Selects Elements by Class Name

📌 **Example:**

```
<p class="text">First paragraph</p>

<p class="text">Second paragraph</p>

let paragraphs = document.getElementsByClassName("text");

console.log(paragraphs[0].innerHTML);  // Output: First paragraph
```

✓ Returns a **collection** of elements with the class "text".

2.3 getElementsByTagName() – Selects Elements by Tag Name

📌 **Example:**

```
let allParagraphs = document.getElementsByTagName("p");

console.log(allParagraphs.length);  // Output: Total number of <p> elements
```

2.4 querySelector() – Selects the First Matching Element

📌 **Example:**

```
let firstParagraph = document.querySelector(".text");

console.log(firstParagraph.innerHTML);  // Output: First paragraph
```

2.5 querySelectorAll() – Selects All Matching Elements

📌 **Example:**

```
let allParagraphs = document.querySelectorAll(".text");

console.log(allParagraphs.length);  // Output: Number of elements with class "text"
```

CHAPTER 3: CHANGING ELEMENTS IN THE DOM

3.1 Changing Text Content

📌 **Example:**

```
document.getElementById("title").innerHTML = "Welcome to JavaScript!";
```

✓ Updates the **content** of the <h1> tag.

3.2 Changing CSS Styles

📌 **Example:**

```
document.getElementById("title").style.color = "blue";

document.getElementById("title").style.fontSize = "24px";
```

✓ Modifies the **color** and **size** of the heading.

3.3 Changing Attributes (e.g., Image Source)

📌 **Example:**

```
<img id="myImage" src="old.jpg">

document.getElementById("myImage").src = "new.jpg";
```

✓ Changes the **image source dynamically**.

3.4 Adding and Removing Classes

📌 **Example:**

```
document.getElementById("title").classList.add("highlight");  // Add class

document.getElementById("title").classList.remove("highlight");  // Remove class
```

✓ Helps toggle **CSS classes** dynamically.

CHAPTER 4: EVENT HANDLING IN JAVASCRIPT

4.1 What are Events?

Events are **user interactions** such as:

✓ Clicking a button

✓ Typing in an input field

✓ Moving the mouse

4.2 Adding Event Listeners

JavaScript can detect and respond to events using addEventListener().

📌 **Example: Button Click Event**

```
<button id="myButton">Click Me</button>

<p id="message"></p>

document.getElementById("myButton").addEventListener("click", function() {

  document.getElementById("message").innerHTML = "Button Clicked!";

});
```

✓ Clicking the button changes the paragraph content.

4.3 Common JavaScript Events

Event	Description
click	When an element is clicked

mouseover	When the mouse hovers over an element
mouseout	When the mouse leaves an element
keydown	When a key is pressed
keyup	When a key is released
change	When the value of an input field changes

📌 **Example: Changing Background on Mouseover**

```
<div id="box" style="width:100px; height:100px;
background:red;"></div>

document.getElementById("box").addEventListener("mouseover",
function() {

  this.style.backgroundColor = "green";

});
```

✓ Hovering over the box changes its color to green.

4.4 Handling Form Events

📌 **Example: Form Validation**

```
<input type="text" id="username" placeholder="Enter your name">

<button id="submitBtn">Submit</button>

<p id="error"></p>

document.getElementById("submitBtn").addEventListener("click",
function() {
```

```
  let name = document.getElementById("username").value;

  if (name === "") {

    document.getElementById("error").innerHTML = "Name cannot be empty!";

  } else {

    document.getElementById("error").innerHTML = "Form submitted!";

  }

});
```

✓ If the input is empty, an error message appears.

CHAPTER 5: EXERCISE

5.1 Multiple Choice Questions

1. Which method selects an element by its id?
   - (a) document.querySelector()
   - (b) document.getElementById() ✅
   - (c) document.getElementsByClassName()
   - (d) document.querySelectorAll()
2. Which event fires when a user **clicks** an element?
   - (a) keydown
   - (b) mouseover

- (c) click ✅
- (d) input

3. What does document.getElementById("title").innerHTML = "New Title"; do?
   - (a) Changes the **font size** of the title
   - (b) Changes the **content** of the element ✅
   - (c) Adds a **new element**
   - (d) Deletes the element

5.2 Practical Tasks

📌 **Task 1:** Create a button that **changes the background color** of the page when clicked.

📌 **Task 2:** Write JavaScript to **hide and show** a paragraph when clicking a button.

📌 **Task 3:** Create a form with an **input field and submit button**. If the input is empty, display an error message.

CHAPTER 6: SUMMARY

✅ The **DOM (Document Object Model)** represents webpage elements in a structured way.
✅ JavaScript can **select, modify, and style** HTML elements dynamically.
✅ **Event handling** allows JavaScript to respond to user interactions.

✅ The addEventListener() method attaches event handlers to elements.

✅ JavaScript helps create **interactive and dynamic** webpages.

☑ Creating Simple Web-Based Games (e.g., Rock-Paper-Scissors)

Chapter 1: Introduction to Web-Based Games

1.1 What are Web-Based Games?

Web-based games are games that are played through a web browser, typically written in **HTML**, **CSS**, and **JavaScript**. These games are interactive and dynamic, allowing users to play directly on websites without needing to download any special software.

1.2 Benefits of Creating Simple Web-Based Games

Creating web-based games: ✓ Enhances **JavaScript** skills.
✓ Introduces basic **game logic** and **user interaction**.
✓ Makes use of **HTML elements** and **CSS for styling**.
✓ Offers a fun and interactive way to learn programming.

Chapter 2: Understanding the Rock-Paper-Scissors Game

2.1 What is Rock-Paper-Scissors?

Rock-Paper-Scissors is a simple hand game played between two people. Each player simultaneously forms one of three shapes with their hand:

- **Rock** (🪨)
- **Paper** (📄)
- **Scissors** (✂️)

The rules are:

- **Rock** crushes **Scissors**.
- **Scissors** cuts **Paper**.
- **Paper** covers **Rock**.

2.2 Goal of the Game

The goal is to determine who wins each round by comparing the chosen shapes:

- **Rock vs Scissors**: Rock wins.
- **Scissors vs Paper**: Scissors wins.
- **Paper vs Rock**: Paper wins.

CHAPTER 3: SETTING UP THE GAME WITH HTML

3.1 Structure of the Web Page

Create an HTML file for the game's basic structure. It will contain:

- A **title** for the game.
- A **set of buttons** for the player to choose rock, paper, or scissors.
- A **message area** to show the result (win, lose, or tie).

📌 **Code (index.html):**

```
<!DOCTYPE html>

<html lang="en">

<head>
```

```
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Rock Paper Scissors</title>
  <link rel="stylesheet" href="style.css">  <!-- Linking CSS file -->
</head>
<body>
  <h1>Rock Paper Scissors Game</h1>
  <div id="choices">
    <button onclick="playGame('rock')">Rock</button>
    <button onclick="playGame('paper')">Paper</button>
    <button onclick="playGame('scissors')">Scissors</button>
  </div>
  <p id="result"></p>
  <script src="script.js"></script>  <!-- Linking JavaScript file -->
</body>
</html>
```

✓ Explanation:

- **Buttons** are provided for the user to select rock, paper, or scissors.
- **JavaScript** will handle the logic behind the game and interact with the webpage.

CHAPTER 4: STYLING THE GAME WITH CSS

4.1 Basic CSS to Style the Page

Add basic styling to make the game visually appealing.

📌 **Code (style.css):**

```
body {
  font-family: Arial, sans-serif;
  text-align: center;
  background-color: #f4f4f4;
  padding: 20px;
}

h1 {
  color: #333;
}

#choices button {
  font-size: 18px;
  padding: 10px 20px;
  margin: 10px;
  cursor: pointer;
```

```
}


#result {
  font-size: 20px;
  font-weight: bold;
}
```

✓ **Explanation:**

- The **button styling** makes the game interface more user-friendly.
- The **result text** will stand out by increasing its font size and weight.

CHAPTER 5: ADDING GAME LOGIC WITH JAVASCRIPT

5.1 Handling Player's Choice

The game logic is implemented using JavaScript. We will create a function playGame() to handle the logic for:

1. Getting the player's choice.
2. Generating a random choice for the computer.
3. Comparing the two choices to determine the winner.

📌 **Code (script.js):**

```
function playGame(playerChoice) {
  // Computer's choices
```

```
const choices = ['rock', 'paper', 'scissors'];
const computerChoice = choices[Math.floor(Math.random() * 3)];

// Display the player's and computer's choices
console.log("Player Choice: " + playerChoice);
console.log("Computer Choice: " + computerChoice);

// Determine the winner
let result;
if (playerChoice === computerChoice) {
  result = "It's a tie!";
} else if (
  (playerChoice === 'rock' && computerChoice === 'scissors') ||
  (playerChoice === 'scissors' && computerChoice === 'paper') ||
  (playerChoice === 'paper' && computerChoice === 'rock')
) {
  result = "You win!";
} else {
  result = "You lose!";
}
```

```
  // Show the result
  document.getElementById("result").innerHTML = result;
}
```

✓ **Explanation:**

- The **computerChoice** is randomly generated using Math.random().
- The function compares the player's and computer's choices and displays the result.
- The **result** is displayed in the <p> element with the ID "result".

CHAPTER 6: ENHANCEMENTS AND FURTHER IDEAS

6.1 Adding Score Tracking

You can track the score to see how many games the player wins, loses, or ties.

📌 **Code (Add score tracking to script.js):**

```
let playerScore = 0;
let computerScore = 0;
let ties = 0;

function playGame(playerChoice) {
  const choices = ['rock', 'paper', 'scissors'];
  const computerChoice = choices[Math.floor(Math.random() * 3)];
```

```
  let result;
  if (playerChoice === computerChoice) {
    result = "It's a tie!";
    ties++;
  } else if (
    (playerChoice === 'rock' && computerChoice === 'scissors') ||
    (playerChoice === 'scissors' && computerChoice === 'paper') ||
    (playerChoice === 'paper' && computerChoice === 'rock')
  ) {
    result = "You win!";
    playerScore++;
  } else {
    result = "You lose!";
    computerScore++;
  }

  document.getElementById("result").innerHTML = result +
"<br>Player Score: " + playerScore + " | Computer Score: " +
computerScore + " | Ties: " + ties;
}
```

✓ **Effect:** Now the webpage will display the **current score** (player, computer, ties).

6.2 Improving User Experience (UX)

To make the game more engaging:

1. Add **animations** when a player selects an option.
2. Change button colors or add **sound effects** when an option is clicked.
3. Display a **reset button** to restart the game and clear the score.

📌 **Code for Reset Button (Add to HTML):**

```
<button onclick="resetGame()">Reset Game</button>
```

📌 **Code for resetGame() function (script.js):**

```
function resetGame() {
  playerScore = 0;
  computerScore = 0;
  ties = 0;
  document.getElementById("result").innerHTML = "Game has been reset.";
}
```

CHAPTER 7: EXERCISE

7.1 Multiple Choice Questions

1. How does the computer generate its choice in the game?
    - (a) Randomly using Math.random() ✅
    - (b) By selecting based on the time of day
    - (c) Based on the player's previous choice
    - (d) It always chooses "rock"
2. What does the playGame() function return when there's a tie?
    - (a) "You win!"
    - (b) "You lose!"
    - (c) "It's a tie!" ✅
    - (d) "Game Over"

7.2 Practical Tasks

📌 **Task 1: Enhance the game to display the computer's choice dynamically.**
📌 **Task 2: Modify the score system to allow resetting the score after each game.**
📌 **Task 3: Add an image for each choice (rock, paper, scissors) and display them when selected.**

CHAPTER 8: SUMMARY

✅ **Web-based games** like Rock-Paper-Scissors are an excellent way to practice **JavaScript** and **DOM manipulation**.

✅ **DOM manipulation** allows you to interact with and change elements of the webpage dynamically based on user actions.

✅ JavaScript handles **events** such as clicks, which are essential for interactivity.

✅ Enhancements like **score tracking, reset buttons**, and **animations** can improve the overall user experience.

☑ CONNECTING HTML, CSS & JAVASCRIPT FOR DYNAMIC WEBSITES

CHAPTER 1: INTRODUCTION TO INTEGRATING HTML, CSS, AND JAVASCRIPT

1.1 What are HTML, CSS, and JavaScript?

To create a dynamic and interactive website, we need to use the three core technologies:

- **HTML (HyperText Markup Language):** Provides the **structure** of a webpage. It uses tags (e.g., , <p>, <h1>) to define elements and their content.
- **CSS (Cascading Style Sheets):** Defines the **style** and appearance of HTML elements (e.g., colors, fonts, spacing). It controls the layout and design of the webpage.
- **JavaScript:** Adds **interactivity** to the webpage. It allows you to manipulate HTML elements, respond to user actions, and handle logic such as form validation, animations, and data fetching.

When combined, these technologies form the **foundation of web development**.

CHAPTER 2: HOW HTML, CSS, AND JAVASCRIPT WORK TOGETHER

2.1 The Relationship Between HTML, CSS, and JavaScript

- **HTML** provides the content and structure of the webpage.

- **CSS** is used to **style** the structure provided by HTML, making the page look good.
- **JavaScript** interacts with the HTML and CSS to add **dynamic behaviors**.

These technologies work together seamlessly to provide a complete and interactive web experience.

2.2 How They Interact: A Simple Example

Consider the following simple example: we have a webpage that allows the user to click a button to change the background color dynamically.

CHAPTER 3: EXAMPLE OF CONNECTING HTML, CSS, AND JAVASCRIPT

3.1 Creating a Simple Webpage with Interactivity

1. HTML – Defining the Structure

The HTML file provides the structure and elements on the webpage.

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

```
  <title>Dynamic Webpage</title>
  <link rel="stylesheet" href="style.css">  <!-- Linking CSS file -->
</head>
<body>
  <h1>Click the Button to Change the Background Color</h1>
  <button id="colorButton">Change Color</button>
  <script src="script.js"></script>  <!-- Linking JavaScript file -->
</body>
</html>
```

- **HTML tags** such as <h1> and <button> are used to create the content.
- The id="colorButton" attribute allows us to refer to the button in JavaScript.
- The **CSS file** and **JavaScript file** are linked to the HTML file using <link> and <script>, respectively.

2. CSS – Styling the Page

Now, let's style the page with some basic CSS. The CSS file defines the look and feel of the webpage.

```
/* style.css */
body {
  font-family: Arial, sans-serif;
  text-align: center;
```

```
    background-color: lightblue;
    padding: 50px;
}

h1 {
    color: darkblue;
}

button {
    padding: 10px 20px;
    font-size: 18px;
    color: white;
    background-color: darkblue;
    border: none;
    cursor: pointer;
}

button:hover {
    background-color: blue;
}
```

- The **background color** is set to **lightblue,** and the **button** is styled to look better and become interactive when hovered over.

3. JavaScript – Adding Interactivity

Now, we need to add the functionality that changes the background color when the button is clicked.

```
// script.js

document.getElementById("colorButton").addEventListener("click", function() {

  document.body.style.backgroundColor = "green";

});
```

- The getElementById() method retrieves the button by its id ("colorButton").
- The addEventListener() method listens for the click event, triggering the function that changes the background color of the body.

3.2 Explanation of How HTML, CSS, and JavaScript Work Together

1. **HTML** creates the structure: an <h1> for the header and a <button> to interact with.
2. **CSS** styles the elements: it defines the colors, fonts, and layout of the webpage.

3. **JavaScript** makes the page dynamic: it listens for user interaction (button click) and modifies the page by changing the background color.

CHAPTER 4: BEST PRACTICES FOR COMBINING HTML, CSS, AND JAVASCRIPT

4.1 Keeping Code Organized

- **HTML** should only handle the content and structure of the page.
- **CSS** should only be used for styling and layout.
- **JavaScript** should only be responsible for interactivity and behavior.
- **Avoid Inline JavaScript and CSS** (e.g., <style> and <script> within the HTML file). Instead, use separate files for each technology.

4.2 Using External Files

To keep your code clean and maintainable, you should always use external files for CSS and JavaScript.

Example Directory Structure:

/project-folder

/index.html

/style.css

/script.js

This structure ensures that your files are well-organized and easy to maintain.

4.3 Modifying HTML with JavaScript

JavaScript can interact with and manipulate HTML elements dynamically. Here are some common DOM manipulation methods:

- **Changing text content:**
- document.getElementById("myText").innerText = "New Text!";
- **Changing attributes (e.g., image source):**
- document.getElementById("myImage").src = "newImage.jpg";
- **Adding or removing classes:**
- document.getElementById("myElement").classList.add("newClass");
- document.getElementById("myElement").classList.remove("oldClass");
- **Creating new elements:**
- let newDiv = document.createElement("div");
- newDiv.innerText = "Hello, World!";
- document.body.appendChild(newDiv);

CHAPTER 5: EXERCISES

5.1 Multiple Choice Questions

1. Which HTML tag is used to include a CSS file?
    - (a) <link> ✅
    - (b) <style>
    - (c) <script>
    - (d) <head>
2. What does the addEventListener() method do in JavaScript?
    - (a) It attaches an event handler to an element. ✅
    - (b) It adds a class to an element.
    - (c) It changes the HTML content of an element.
    - (d) It removes an element from the DOM.
3. How can you change the background color of the body in JavaScript?
    - (a) document.body.style.backgroundColor = "blue"; ✅
    - (b) document.body.bgColor = "blue";
    - (c) document.style.backgroundColor = "blue";
    - (d) document.getElementById("body").style.backgroundColor = "blue";

5.2 Practical Tasks

📌 **Task 1:** Create a webpage with a button that changes the text color when clicked.

📌 **Task 2:** Write JavaScript code that changes the background color

every time the mouse hovers over a div.

📌 **Task 3:** Create a simple form that takes user input and displays a welcome message with JavaScript.

CHAPTER 6: SUMMARY

✅ **HTML** structures the webpage with elements such as text, buttons, and images.

✅ **CSS** styles the elements, making them visually appealing.

✅ **JavaScript** adds interactivity and modifies HTML/CSS based on user actions.

✅ Keep code organized by using **external files** for CSS and JavaScript.

✅ JavaScript's **DOM manipulation** methods allow dynamic changes to the webpage.

ASSIGNMENT 1:

DEVELOP A SIMPLE TO-DO LIST APPLICATION USING JAVASCRIPT.

Assignment Solution 1: Develop a Simple To-Do List Application Using JavaScript

🎯 Objective:

✓ Create an interactive **To-Do List** using JavaScript.
✓ Learn to handle **user inputs** and **dynamic updates** of content.
✓ Use **HTML** for structure, **CSS** for styling, and **JavaScript** for functionality.

🛠 Step-by-Step Guide

Step 1: Set Up Your Project Folder

1. Create a new folder, e.g., **"todo-app"**.
2. Inside the folder, create three files:
   - **index.html** – HTML file for the structure of the webpage.
   - **style.css** – CSS file for styling the webpage.
   - **script.js** – JavaScript file for the to-do list functionality.

Step 2: Create the Basic HTML Structure

Your **index.html** file will contain the basic structure for the To-Do List application.

Code (index.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>
  <LINK REL="STYLESHEET" HREF="STYLE.CSS"> <!-- LINKING CSS -->
</HEAD>
<BODY>
  <DIV CLASS="CONTAINER">
    <H1>MY TO-DO LIST</H1>
    <INPUT TYPE="TEXT" ID="NEWTASK" PLACEHOLDER="ADD A NEW TASK">
    <BUTTON ONCLICK="ADDTASK()">ADD TASK</BUTTON>

    <UL ID="TASKLIST">
      <!-- TO-DO TASKS WILL APPEAR HERE -->
    </UL>
  </DIV>

  <SCRIPT SRC="SCRIPT.JS"></SCRIPT> <!-- LINKING JAVASCRIPT -->
```

```
</BODY>

</HTML>
```

✓ Explanation:

- The <input> field is where the user will enter new tasks.
- The <button> will call the addTask() function to add the task.
- The <ul> element with the ID taskList will display the list of tasks.

Step 3: Style the Application with CSS

Now, let's add some basic styling to make the app look better.

📌 Code (style.css):

```
BODY {
  FONT-FAMILY: ARIAL, SANS-SERIF;
  BACKGROUND-COLOR: #F4F4F4;
  MARGIN: 0;
  PADDING: 0;
}

.CONTAINER {
  MAX-WIDTH: 600PX;
  MARGIN: 0 AUTO;
  PADDING: 20PX;
```

```
  BACKGROUND-COLOR: WHITE;
  BORDER-RADIUS: 8PX;
  BOX-SHADOW: 0PX 0PX 10PX RGBA(0, 0, 0, 0.1);
}

H1 {
  TEXT-ALIGN: CENTER;
  COLOR: #333;
}

#NEWTASK {
  WIDTH: 80%;
  PADDING: 10PX;
  MARGIN: 10PX 0;
  BORDER-RADIUS: 5PX;
  BORDER: 1PX SOLID #CCC;
}

BUTTON {
  PADDING: 10PX;
  WIDTH: 15%;
  BACKGROUND-COLOR: #4CAF50;
```

```
  COLOR: WHITE;
  BORDER: NONE;
  BORDER-RADIUS: 5PX;
  CURSOR: POINTER;
}

BUTTON:HOVER {
  BACKGROUND-COLOR: #45A049;
}

UL {
  LIST-STYLE-TYPE: NONE;
  PADDING: 0;
}

LI {
  DISPLAY: FLEX;
  JUSTIFY-CONTENT: SPACE-BETWEEN;
  PADDING: 10PX;
  BACKGROUND-COLOR: #F9F9F9;
  MARGIN: 5PX 0;
  BORDER-RADIUS: 5PX;
```

```
}

LI.COMPLETED {
  TEXT-DECORATION: LINE-THROUGH;
  BACKGROUND-COLOR: #D3FFD3;
}

BUTTON.DELETE {
  BACKGROUND-COLOR: #F44336;
  COLOR: WHITE;
  BORDER: NONE;
  BORDER-RADIUS: 5PX;
  CURSOR: POINTER;
}

BUTTON.DELETE:HOVER {
  BACKGROUND-COLOR: #DA190B;
}
```

✓ EXPLANATION:

- STYLED THE **INPUT FIELD, BUTTON,** AND **LIST ITEMS** FOR A CLEAN AND SIMPLE DESIGN.
- THE LI.COMPLETED CLASS APPLIES A **LINE-THROUGH** TO MARK COMPLETED TASKS.

- The delete button is styled to appear red with a hover effect.

Step 4: Add Functionality with JavaScript

Next, we'll write the JavaScript code to handle adding, completing, and deleting tasks.

📌 **Code (script.js):**

```
// Function to add a new task
function addTask() {
  let taskInput = document.getElementById("newTask");
  let taskText = taskInput.value.trim();

  if (taskText !== "") {
    // Create new list item
    let li = document.createElement("li");

    // Add task text to the list item
    li.textContent = taskText;

    // Create a delete button
    let deleteButton = document.createElement("button");
    deleteButton.textContent = "Delete";
    deleteButton.className = "delete";
```

```
    // Append the delete button to the list item
    li.appendChild(deleteButton);

    // Add event listener to delete the task
    deleteButton.addEventListener("click", function() {
      li.remove();
    });

    // Add event listener to mark the task as completed
    li.addEventListener("click", function() {
      li.classList.toggle("completed");
    });

    // Add the new task to the list
    document.getElementById("taskList").appendChild(li);

    // Clear the input field
    taskInput.value = "";
  } else {
    alert("Please enter a task!");
  }
}
```

✓ Explanation:

- **addTask() function**:
  - Retrieves the task text from the input field.
  - If the input is not empty, it creates a new li element and adds it to the list.
  - A **delete button** is created for each task, and clicking it removes the task from the list.
  - Clicking on a task will **mark it as completed**, toggling the "completed" class.
  - After adding the task, the input field is cleared.

Step 5: Test the Application

1. **Save all files** (index.html, style.css, and script.js).
2. Open **index.html** in a browser.
3. Test the app:
   - **Add tasks** by typing in the input field and clicking "Add Task."
   - **Mark tasks as completed** by clicking on them.
   - **Delete tasks** by clicking the "Delete" button.

Step 6: Enhance the To-Do List Application

Here are some enhancements you can add:

1. **Save the tasks** in **localStorage** so that the tasks persist even after the browser is closed.

2. Add a **Task Priority** feature, where tasks can be marked as high, medium, or low priority.
3. Implement a **Clear All Tasks** button to remove all tasks from the list.
4. Improve the **UX/UI** by adding icons for the delete button, or animating the task completion.

🎯 Summary

☑ **HTML** provides the structure of the To-Do list (input fields, buttons, list).

☑ **CSS** is used to style the list, buttons, and form inputs, making it visually appealing.

☑ **JavaScript** adds interactivity, allowing tasks to be added, marked as completed, or deleted.

☑ This simple To-Do list application demonstrates the basic functionality and interaction between **HTML, CSS, and JavaScript**.

ASSIGNMENT 2:

CREATE AN INTERACTIVE QUIZ APPLICATION WITH SCORE TRACKING.

💡 Assignment Solution 2: Create an Interactive Quiz Application with Score Tracking

---

🎯 Objective:

✓ Create a **quiz application** using **HTML** for structure, **CSS** for styling, and **JavaScript** for interactivity and score tracking.
✓ Learn to **handle user input, display questions**, and **calculate scores** dynamically.

🛠 Step-by-Step Guide

Step 1: Set Up Your Project Folder

1. Create a new folder, e.g., **"quiz-app"**.
2. Inside the folder, create three files:
    - **index.html** – HTML file for the structure of the webpage.
    - **style.css** – CSS file for styling.
    - **script.js** – JavaScript file for the quiz functionality.

Step 2: Create the Basic HTML Structure

The **index.html** file will contain the structure and layout of the quiz.

📌 **Code (index.html):**

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Interactive Quiz</title>
  <link rel="stylesheet" href="style.css"> <!-- Linking CSS -->
</head>
<body>
  <div class="quiz-container">
    <h1>Interactive Quiz</h1>
    <div id="quiz">
      <!-- Quiz questions will be inserted here dynamically -->
    </div>
    <button id="nextButton" onclick="nextQuestion()">Next Question</button>
    <div id="score">Score: 0</div>
  </div>

  <script src="script.js"></script> <!-- Linking JavaScript -->
</body>
</html>
```

✓ Explanation:

- The page contains an empty div with id quiz where the quiz questions will be dynamically inserted using JavaScript.
- The button "Next Question" will be used to move to the next question.
- The current **score** is displayed in the div with id score.

Step 3: Style the Quiz with CSS

Now, let's add some basic styling to make the quiz look more appealing.

📌 **Code (style.css):**

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f9;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  margin: 0;
}

.quiz-container {
  text-align: center;
  background-color: white;
```

```
    padding: 30px;
    border-radius: 8px;
    box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
    width: 300px;
}

h1 {
    font-size: 24px;
    margin-bottom: 20px;
}

button {
    padding: 10px 20px;
    font-size: 16px;
    margin-top: 20px;
    cursor: pointer;
    background-color: #4CAF50;
    color: white;
    border: none;
    border-radius: 5px;
}
```

```
button:hover {
  background-color: #45a049;
}

#score {
  font-size: 18px;
  margin-top: 15px;
}
```

✓ **Explanation:**

- The **quiz container** is centered on the page with some padding, a background color, and a shadow to make it stand out.
- Buttons are styled to be large and easy to click, with hover effects.
- The **score** text is styled to stand out clearly.

Step 4: Add JavaScript for Quiz Functionality

Now, let's add the logic for displaying questions, tracking the score, and navigating through the quiz.

📌 **Code (script.js):**

```
let currentQuestionIndex = 0;
let score = 0;
```

```
const questions = [
  {
    question: "What is the capital of France?",
    options: ["Berlin", "Madrid", "Paris", "Lisbon"],
    correctAnswer: "Paris"
  },
  {
    question: "What is the largest planet in our solar system?",
    options: ["Earth", "Mars", "Jupiter", "Saturn"],
    correctAnswer: "Jupiter"
  },
  {
    question: "Which programming language is known as the language of the web?",
    options: ["Python", "Java", "JavaScript", "C++"],
    correctAnswer: "JavaScript"
  }
];

// Function to display the current question
function displayQuestion() {
  const currentQuestion = questions[currentQuestionIndex];
  const quizContainer = document.getElementById("quiz");
```

```
  // Create the question text
  quizContainer.innerHTML = `
    <h2>${currentQuestion.question}</h2>
    <ul>
      ${currentQuestion.options.map(option => `<li><button
onclick="checkAnswer('${option}')">${option}</button></li>`).join('')
}
    </ul>
  `;
}

// Function to check if the selected answer is correct
function checkAnswer(selectedAnswer) {
  const currentQuestion = questions[currentQuestionIndex];

  if (selectedAnswer === currentQuestion.correctAnswer) {
    score++;
  }

  // Update the score on the page
  document.getElementById("score").innerText = `Score: ${score}`;
```

```
    document.getElementById("nextButton").style.display = "inline-block";  // Show the Next button after answering
}

// Function to go to the next question
function nextQuestion() {
    currentQuestionIndex++;

    // Hide the Next button until the user answers the next question
    document.getElementById("nextButton").style.display = "none";

    if (currentQuestionIndex < questions.length) {
        displayQuestion();
    } else {
        // Show the final score when all questions are answered
        const quizContainer = document.getElementById("quiz");
        quizContainer.innerHTML = `<h2>Quiz Complete!</h2><p>Your final score is: ${score}/${questions.length}</p>`;
        document.getElementById("nextButton").style.display = "none"; // Hide Next button
    }
}
```

```
// Display the first question when the page loads
displayQuestion();
```

✓ **Explanation:**

- **questions array:** Contains the quiz questions, options, and the correct answer.
- **displayQuestion():** This function dynamically displays the question and its options as buttons.
- **checkAnswer():** When the user selects an answer, this function checks if it's correct and updates the score.
- **nextQuestion():** This function is called when the user clicks the "Next Question" button, advancing to the next question or showing the final score if the quiz is complete.

Step 5: Test the Application

1. **Save all files** (index.html, style.css, and script.js).
2. Open **index.html** in a browser.
3. Try answering the quiz questions:
    - Select an answer, and the score will be updated.
    - Click the "Next Question" button to proceed.
    - Once all questions are answered, the final score will be displayed.

Step 6: Enhance the Quiz Application

Here are some enhancements you can add:

1. **Shuffle the options** for each question to make the quiz more challenging.
2. Add a **timer** to the quiz to track how long it takes to complete.
3. Implement a **reset button** to restart the quiz.
4. Make the quiz **responsive** for mobile devices using media queries.

🎯 Summary

✅ **HTML** provides the structure for displaying questions, buttons, and the score.
✅ **CSS** is used to style the quiz elements for a clean, user-friendly interface.
✅ **JavaScript** handles the quiz logic, including displaying questions, checking answers, and updating the score.
✅ **Event listeners** in JavaScript are used to interact with user actions, like selecting answers and navigating between questions.