**Module-6(Mernstack)**

**JavaScript Essential and Advanced**

* **JavaScript Introduction**

(*Theory Assignment*)

Question 1: What is JavaScript? Explain the role of JavaScript in web

development.

Ans:

* JavaScript is a high-level, lightweight, and interpreted programming language primarily used to make web pages interactive and dynamic. It is one of the core technologies of the web, along with HTML (for structure) and CSS (for styling).
* **Uses:**
* Manipulates the DOM to update content in real-time
* Handles user events (clicks, inputs, etc.)
* Validates forms and reduces server load
* Builds interactive UIs and modern web apps
* **Frameworks & Libraries:**
* Popular ones include React.js, Angular, Vue.js for front-end, and Node.js, Express.js for back-end.
* **Advantages:**
* Runs directly in browsers (no installation needed)
* Cross-platform and versatile (front-end + back-end)
* Huge community support and ready-to-use libraries
* Makes websites fast, interactive, and user-friendly
* **Role of JavaScript in Web Development:**

1. **Interactivity:** JavaScript allows developers to add interactive features such as image sliders, dropdown menus, form validation, popups, and dynamic content updates without reloading the page.
2. **DOM Manipulation:** It can access and modify the HTML and CSS of a webpage in real-time, enabling changes like updating text, styles, or structure dynamically.
3. **Event Handling:** JavaScript responds to user actions such as clicks, key presses, or mouse movements, improving user experience.
4. **Client-Side Scripting:** It reduces server load by validating inputs and performing calculations directly in the browser.
5. **Modern Web Applications:** With frameworks and libraries (e.g., React, Angular, Vue), JavaScript powers Single Page Applications (SPAs) and complex front-end interfaces.
6. **Server-Side Development:** Using environments like **Node.js**, JavaScript can also be used for back-end development, making it a full-stack language.

Question 2: How is JavaScript different from other programming

languages like Python or Java?

Ans:

* JavaScript, Python, and Java are all powerful programming languages, but they serve different purposes and have unique characteristics.

1. **Purpose and Usage:**
   * JavaScript is mainly used for web development, adding interactivity, animations, and dynamic behavior to websites. With Node.js, it can also be used on the server-side.
   * Python is a general-purpose language widely used in data science, AI/ML, automation, and backend development.
   * Java is a strongly typed, object-oriented language, mainly used in enterprise applications, Android app development, and large-scale backend systems.
2. **Execution Environment:**
   * JavaScript runs natively in web browsers, making it the “language of the web.”
   * Python requires an interpreter (e.g., CPython) to execute.
   * Java code is compiled into bytecode and runs on the Java Virtual Machine (JVM).
3. **Typing System:**
   * JavaScript is dynamically typed, giving flexibility but sometimes causing runtime errors.
   * Python is also dynamically typed, but emphasizes readability and simplicity.
   * Java is statically typed, requiring variable types to be defined, which improves performance and reliability in large applications.
4. **Performance:**
   * Java generally provides the best performance due to compilation and strong typing.
   * JavaScript is fast for web-related tasks, especially with modern engines like Google’s V8, but can be less efficient for heavy computation.
   * Python is slower compared to Java and JavaScript, but its vast libraries make it highly productive.
5. **Ecosystem and Frameworks:**
   * **JavaScript:** React, Angular, Vue (frontend); Node.js, Express (backend).
   * **Python:** Django, Flask (web); TensorFlow, PyTorch (AI/ML).
   * **Java:** Spring, Hibernate (enterprise apps); Android SDK (mobile apps).

* **Conclusion:**  
  JavaScript is the backbone of interactive web development, Python excels in data science and AI, and Java is dominant in enterprise and mobile application development. Each has its strengths, and the choice depends on the project requirements.

Question 3: Discuss the use of <script> tag in HTML. How can you link

an external JavaScript file to an HTML document?

Ans:

* ***Use of <script> Tag in HTML :***

The <script> tag in HTML is used to embed JavaScript code or to link an external JavaScript file into a webpage. It enables us to add interactivity, dynamic content, form validation, and other client-side functionalities.

* If written inside HTML, the code goes between <script> ... </script> tags.
* It can be placed inside the <head> or at the end of the <body> section. Placing it at the end improves page loading speed because scripts load after the content.
* ***Linking an External JavaScript File :***
* To keep code organized, JavaScript is often written in a separate file with the .js extension. We link it using the **src attribute** of the <script> tag.
* Syntax:
  + <script src="script.js"></script>

Here, script.js is the external JavaScript file.

This tag should be placed before the closing </body> tag so that the HTML loads first, then the script runs.

(*Lab Assignment*)

Task:

• Create a simple HTML page and add a <script> tag within the page.

• Write JavaScript code to display an alert box with the message "Welcome to

JavaScript!" when the page loads.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>JavaScript Alert Example</title>

</head>

<body>

<h1>Hello, HTML with JavaScript</h1>

<! -- JavaScript code inside <script> tag -->

<script>

// This code runs when the page loads

alert ("Welcome to JavaScript!");

</script>

</body>

</html>

* **Variables and Data Types**

(*Theory Assignment*)

Question 1: What are variables in JavaScript? How do you declare a

variable using var, let, and const?

Ans:

* In JavaScript, a **variable** is a named container used to store data values, such as numbers, strings, arrays, objects, or functions. Variables make programs dynamic because we can reuse and manipulate values during execution.
* Rules for Naming Variables:
* Variable names must begin with a letter, underscore (\_), or dollar sign ($).
* They cannot start with a number.
* Names are case-sensitive (age and Age are different)
* Reserved keywords (like let, class, return) cannot be used as variable names.
* ***Declaring Variables in JavaScript:***

JavaScript provides three ways to declare variables:

**1) var**

* The oldest way (before ES6).
* Function-scoped → accessible within the function where declared.
* Can be redeclared and updated.
* Hoisted (moved to the top of scope), but initialized with undefined.
* Example:

var city = "Delhi";

var city = "Mumbai”; // Redeclaration allowed

city = "Chennai”; // Update allowed

**2) let**

* Introduced in ES6 (2015).
* Block-scoped → only accessible inside { }.
* Can be updated but not redeclared in the same scope.
* Not hoisted in the same way as var (gives error if accessed before declaration).
* Example:

let age = 20;

// let age = 25; Error (cannot redeclare)

age = 25; // Update allowed

**3) const**

* Introduced in ES6 (2015).
* Block-scoped like let.
* Must be initialized at declaration.
* Cannot be updated or redeclared.
* Best for values that should remain constant.
* Example:

const PI = 3.1416;

// PI = 3.14; Error (cannot update)

// const PI = 3.15; Error (cannot redeclare)

* Advantages of Variables in JavaScript:
* Provide a way to store and reuse data.
* Improve readability and maintainability of code.
* Allow dynamic programming (values can change at runtime with let/var).
* Support scope control using let and const for better security and reduced errors.
* With const, ensure immutability for fixed values like configuration settings.

Question 2: Explain the different data types in JavaScript. Provide

examples for each.

Ans:

* JavaScript has two main categories of data types: **Primitive** and **Non-Primitive (or Reference)**. These types define the kind of data a variable can hold.

1. **Primitive Data Types**

* Primitive types represent single, simple data values. They are immutable, meaning their value cannot be changed once created (though the variable can be reassigned to a new value). There are seven primitive types:

|  |  |  |
| --- | --- | --- |
| Data Type | Description | Example |
| Number | Represents both integer and floating-point numbers. | let age = 30; let price = 19.99; |
| BigInt | Represents integers with arbitrary precision (numbers larger than 253−1). | const hugeNum = 9007199254740991n; |
| String | Represents textual data, enclosed in single quotes (''), double quotes (""), or backticks (` `). | let name = "Alice"; let message = 'Hello World'; |
| Boolean | Represents a logical entity and can only have two values: true or false. | let isStudent = true; let isOver = false; |
| Undefined | A variable that has been declared but has not yet been assigned a value. | let city; // city is undefined |
| Null | Represents the intentional absence of any object value. It must be explicitly assigned. | let response = null; |
| Symbol | A unique and immutable value often used as the key for an object property to avoid collisions. | const id = Symbol('user-id'); |

1. **Non-Primitive Data Types**

* Non-primitive types are used to store collections of data and more complex entities. Unlike primitive values, variables holding non-primitive values store a reference (or memory address) to where the data is actually located. They are mutable.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Description** | **Example** |
| Object | The most fundamental non-primitive type. It is a collection of key-value pairs (properties). | const person = {firstName:"John", age: 25 }; |
| Array | A special type of object used to store ordered lists of values (elements). | const colors = ["red", "green", "blue"]; |
| Function | A callable object that executes a block of code when invoked. Functions are considered first-class objects in JavaScript. | function greet(name) {return 'Hello, ' + name;} |

Question 3: What is the difference between undefined and null in

JavaScript?

Ans:

* Both undefined and null represent **empty values**, but they are used differently in JavaScript.

**1. undefined**

* Means a **variable has been declared but not assigned** a value.
* It is the **default value** of uninitialized variables, function parameters (if no argument is passed), and missing object properties.
* Type of undefined is "undefined".
* Example:

let x;

console.log(x); // undefined

**2. null**

* Represents an **intentional absence of value**.
* Used by developers to assign a “nothing” or “empty” value to a variable.
* Type of null is "object" (this is a historical bug in JavaScript).
* Example:

let y = null;

console.log(y); // null

* Key Differences:

| **Feature** | **undefined** | **null** |
| --- | --- | --- |
| **Meaning** | Variable declared but not assigned. | Intentional empty value. |
| **Assigned by** | JavaScript engine (default). | Developer (manually assigned). |
| **Type** | "undefined" | "object" (bug in JS). |
| **Usage** | Indicates missing/uninitialized value. | Represents "nothing" or "empty". |

(*Lab Assignment*)

Task:

• Write a JavaScript program to declare variables for different data

types (string, number, boolean, null, and undefined).

• Log the values of the variables and their types to the console

using console.log ().

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Document</title>

</head>

<body>

    <script>

        // Declare variables of different data types

        // String

        let name = "Alice";

        // Number

        let age = 25;

        // Boolean

        let isStudent = true;

        // Null

        let car = null;

        // Undefined

        let city; // Not assigned yet

        // Log values and their types to the console

        console.log ("Value of name:", name, " | Type:", typeof name);

        console.log ("Value of age:", age, " | Type:", typeof age);

        console.log ("Value of isStudent:", isStudent, " | Type:", typeof isStudent);

        console.log ("Value of car:", car, " | Type:", typeof car);

        console.log ("Value of city:", city, " | Type:", typeof city);

    </script>

</body>

</html>

* **JavaScript Operators**

(*Theory Assignment*)

Question 1: What are the different types of operators in JavaScript?

Explain with examples.

• Arithmetic operators

• Assignment operators

• Comparison operators

• Logical operators

Ans:

* Operators are symbols or keywords used to perform operations on values or variables.
* JavaScript supports several types of operators, including arithmetic, assignment, comparison, and logical operators.

1. Arithmetic Operator:

Used to perform **mathematical operations** on numbers.

| **Operator** | **Description** | **Example** | **Output** |
| --- | --- | --- | --- |
| + | Addition | 5 + 3 | 8 |
| - | Subtraction | 10 - 4 | 6 |
| \* | Multiplication | 6 \* 3 | 18 |
| / | Division | 12 / 4 | 3 |
| % | Modulus (Remainder) | 10 % 3 | 1 |
| \*\* | Exponentiation | 2 \*\* 3 | 8 |

* Example:

let a = 10;

let b = 3;

console.log ("Addition: ", a + b); // 13

console.log ("Subtraction: ", a - b); // 7

console.log ("Multiplication: ", a \* b); // 30

console.log ("Division: ", a / b); // 3.3333...

console.log ("Modulus: ", a % b); // 1

console.log ("Exponentiation: ", a \*\* b); // 1000

**2. Assignment Operators**

Used to **assign values** to variables.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| = | Assign value | x = 5 |
| += | Add and assign | x += 3 → x = x + 3 |
| -= | Subtract and assign | x -= 2 → x = x - 2 |
| \*= | Multiply and assign | x \*= 4 → x = x \* 4 |
| /= | Divide and assign | x /= 2 → x = x / 2 |
| %= | Modulus and assign | x %= 3 → x = x % 3 |

* Example:

let x = 5;

x += 3; // x = x + 3 → 8

console.log ("x += 3 →", x);

x -= 2; // x = x - 2 → 6

console.log ("x -= 2 →", x);

x \*= 4; // x = x \* 4 → 24

console.log ("x \*= 4 →", x);

x /= 3; // x = x / 3 → 8

console.log ("x /= 3 →", x);

x %= 5; // x = x % 5 → 3

console.log ("x %= 5 →", x);

**3. Comparison Operators**

Used to **compare two values** and return a Boolean (true or false).

| **Operator** | **Description** | **Example** | **Output** |
| --- | --- | --- | --- |
| == | Equal to | 5 == '5' | true |
| === | Strict equal to (value & type) | 5 === '5' | false |
| != | Not equal to | 5! = 3 | true |
| !== | Strict not equal | 5! == '5' | true |
| > | Greater than | 7 > 4 | true |
| < | Less than | 3 < 8 | true |
| >= | Greater than or equal | 5 >= 5 | true |
| <= | Less than or equal | 4 <= 7 | true |

* Example:

let m = 10;

let n = "10";

console.log("m == n →", m == n); // true (value only)

console.log("m === n →", m === n); // false (value + type)

console.log("m != 5 →", m != 5); // true

console.log("m !== '10' →", m !== '10'); // true

console.log("m > 5 →", m > 5); // true

console.log("m < 20 →", m < 20); // true

console.log("m >= 10 →", m >= 10); // true

console.log("m <= 15 →", m <= 15); // true

**4. Logical Operators**

Used to combine or invert **Boolean values**.

| **Operator** | **Description** | **Example** | **Output** |
| --- | --- | --- | --- |
| && | Logical AND | (true && false) | false |
| || | Logical OR | ` | Logical OR |
| ! | Logical NOT | !true | false |

* Example:

let isAdult = true;

let hasID = false;

console.log("isAdult && hasID →", isAdult && hasID); // false (AND)

console.log("isAdult || hasID →", isAdult || hasID); // true (OR)

console.log("!isAdult →", !isAdult); // false (NOT)

Question 2: What is the difference between == and === in JavaScript?

Ans:

* Both == and === are **comparison operators**, but they behave differently:

**1. == (Equality Operator)**

* Compares only the values of two operands.
* Performs type coercion if the data types are different (converts them to the same type before comparison).
* Example:

let a = 5;

let b = "5";

console.log(a == b); // true (values are equal after type coercion)

**2. === (Strict Equality Operator)**

* Compares both the value and the data type.
* No type coercion occurs. Both value and type must be the same for it to return true.
* Example:

let a = 5;

let b = "5";

console.log(a === b); // false (different data types)

**Key Differences**

| **Feature** | **== (Equality)** | **=== (Strict Equality)** |
| --- | --- | --- |
| **Compares** | Values only | Values + Data types |
| **Type Coercion** | Yes | No |
| **Example** | 5 == "5" → true | 5 === "5" → false |

(*Lab Assignment*)

Task:

• Create a JavaScript program to perform the following:

• Add, subtract, multiply, and divide two numbers using arithmetic operators.

• Use comparison operators to check if two numbers are equal and if

one number is greater than the other.

• Use logical operators to check if both conditions (e.g., a > 10and b < 5) are

true.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Document</title>

</head>

<body>

    <script>

        // Declare two numbers

        let a = 15;

        let b = 3;

        // --- Arithmetic Operators ---

        console.log("Arithmetic Operations:");

        console.log("Addition (a + b):", a + b);         // 18

        console.log("Subtraction (a - b):", a - b);      // 12

        console.log("Multiplication (a \* b):", a \* b);   // 45

        console.log("Division (a / b):", a / b);         // 5

        console.log("Modulus (a % b):", a % b);          // 0

        console.log("Exponentiation (a \*\* b):", a \*\* b); // 3375

        console.log("\n--- Comparison Operators ---");

        // Check if two numbers are equal

        console.log("a == b:", a == b);   // false

        console.log("a === b:", a === b); // false

        // Check if one number is greater than the other

        console.log("a > b:", a > b);     // true

        console.log("a < b:", a < b);     // false

        console.log("\n--- Logical Operators ---");

        // Check if both conditions are true: a > 10 and b < 5

        let condition = (a > 10) && (b < 5);

        console.log("a > 10 && b < 5:", condition); // true

        // Example of OR operator

        console.log("a > 20 || b < 5:", (a > 20) || (b < 5)); // true

        // Example of NOT operator

        console.log("!(a > b):", !(a > b)); // false

    </script>

</body>

</html>

* **Control Flow (If-Else, Switch)**

(*Theory Assignment*)

Question 1: What is control flow in JavaScript? Explain how if-else

statements work with an example.

Ans:

* **Control flow** refers to the order in which the statements or instructions in a program are executed. By default, JavaScript executes code **line by line**, but sometimes we need to make decisions or repeat actions. Control flow statements allow us to **control the execution path** based on conditions or loops.
* Common control flow statements include:
  + If , if-else, if else ladder
  + switch
  + Loops (for, while, do…while)
* **If-Else Statements**
  + The **if-else statement** is used to execute a block of code **only if a specified condition is true**, and optionally execute another block if the condition is false.
  + Syntax:

if (condition) {

// code to execute if condition is true

} else {

// code to execute if condition is false

}

* Example:

let age = 18;

if (age >= 18) {

console.log("You are an adult.");

} else {

console.log("You are a minor.");

}

Question 2: Describe how switch statements work in JavaScript. When

should you use a switch statement instead of if-else?

Ans:

* A **switch statement** is a control flow statement that allows you to **execute different blocks of code based on the value of an expression**. It is an alternative to multiple if-else-if statements, especially when comparing a single variable against many possible values.
* Syntax:

switch(expression) {

case value1:

// code to execute if expression === value1

break;

case value2:

// code to execute if expression === value2

break;

...

default:

// code to execute if none of the cases match

}

* **Key Points:**
  + - The **expression** is evaluated once.
    - Each **case** checks for strict equality (===).
    - The **break** statement prevents the code from running into the next case (fall-through).
    - The **default** case runs if no match is found; it is optional.
* Example

let day = 3;

let dayName;

switch(day) {

case 1:

dayName = "Monday";

break;

case 2:

dayName = "Tuesday";

break;

case 3:

dayName = "Wednesday";

break;

case 4:

dayName = "Thursday";

break;

case 5:

dayName = "Friday";

break;

case 6:

dayName = "Saturday";

break;

case 7:

dayName = "Sunday";

break;

default:

dayName = "Invalid day";

}

console.log("Day:", dayName);

* **When to Use switch Instead of if-else**
  + Use switch when you are comparing a single variable against multiple discrete values.
  + It makes the code cleaner and more readable than multiple if-else-if statements.
  + if-else is better when you have complex conditions involving ranges or multiple variables.

(*Lab Assignment*)

Task 1:

• Write a JavaScript program to check if a number is positive, negative, or zero

using an if-else statement.

Task 2:

• Create a JavaScript program using a switch statement to display the day of

the week based on the user input (e.g., 1 for Monday, 2 for Tuesday, etc.).

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Interactive JavaScript Tasks</title>

</head>

<body>

<h2>Task 1: Check if a number is positive, negative, or zero</h2>

<p id="task1"></p>

<h2>Task 2: Display-day of the week using switch</h2>

<p id="task2"></p>

<script>

// --- Task 1: Positive, Negative, or Zero ---

let num = Number(prompt("Enter a number for Task 1:"));

let result1;

if (num > 0) {

result1 = num + " is positive.";

} else if (num < 0) {

result1 = num + " is negative.";

} else {

result1 = num + " is zero.";

}

document.getElementById("task1").innerText = result1;

// --- Task 2: Day of the Week ---

let day = Number(prompt("Enter a number (1-7) for Task 2 to get the day of the week:"));

let dayName;

switch(day) {

case 1:

dayName = "Monday";

break;

case 2:

dayName = "Tuesday";

break;

case 3:

dayName = "Wednesday";

break;

case 4:

dayName = "Thursday";

break;

case 5:

dayName = "Friday";

break;

case 6:

dayName = "Saturday";

break;

case 7:

dayName = "Sunday";

break;

default:

dayName = "Invalid day";

}

document.getElementById("task2").innerText = "Day: " + dayName;

</script>

</body>

</html>

* **Loops (For, While, Do-While)**

(*Theory Assignment*)

Question 1: Explain the different types of loops in JavaScript (for, while,

do-while). Provide a basic example of each.

Ans:

* Loops are used to **repeat a block of code multiple times** until a specified condition is met. JavaScript provides several types of loops:

**1) for loop**

* + The for loop is used when the number of iterations is **known in advance**.
  + **Syntax:**

for(initialization; condition; increment/decrement) {

// code to execute

}

* + **Example:**

for(let i = 1; i <= 5; i++) {

console.log("For loop iteration:", i);

}

**2) while loop**

* + The while loop repeats a block of code **as long as the condition is true**. The condition is checked **before executing the loop body**.
  + **Syntax:**

while(condition) {

// code to execute

}

* **Example:**

let j = 1;

while(j <= 5) {

console.log("While loop iteration:", j);

j++;

}

**3) do-while loop**

* The do-while loop executes the code **at least once**, and then repeats it **as long as the condition is true**. The condition is checked **after the loop body**.
* **Syntax:**

do {

// code to execute

} while(condition);

* **Example:**

let k = 1;

do {

console.log("Do-while loop iteration:", k);

k++;

} while(k <= 5);

Question 2: What is the difference between a while loop and a do-while

loop?

Ans:

* Both while and do-while loops are used to **repeat code based on a condition**, but they differ in **when the condition is checked**.

| **Feature** | **while Loop** | **do-while Loop** |
| --- | --- | --- |
| **Condition Checked** | Before executing the loop body | After executing the loop body |
| **Execution Guarantee** | May execute 0 times if condition is false | Executes at least once regardless of the condition |
| **Syntax** | while(condition) { ... } | do { ... } while(condition); |
| **Use Case** | When you may not want to run the loop at all if the condition is false | When you want to run the loop at least once |

(*Lab Assignment*)

Task 1:

• Write a JavaScript program using a for loop to print numbers from

1 to 10.

Task 2:

• Create a JavaScript program that uses a while loop to sum all

even numbers between 1 and 20.

Task 3:

• Write a do-while loop that continues to ask the user for input until

they enter a number greater than 10.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>JavaScript Loops Tasks</title>

</head>

<body>

<h2>Task 1: Print numbers from 1 to 10 using for loop</h2>

<h2>Task 2: Sum of even numbers between 1 and 20 using while loop</h2>

<h2>Task 3: Ask user for input until number > 10 using do-while loop</h2>

<script>

// --- Task 1: For loop 1 to 10 ---

let numbers = "";

for(let i = 1; i <= 10; i++) {

numbers += i + " ";

}

document.write("<p>Task 1 Output: " + numbers + "</p>");

// --- Task 2: Sum of even numbers 1 to 20 using while loop ---

let sum = 0;

let j = 1;

while(j <= 20) {

if(j % 2 === 0) {

sum += j;

}

j++;

}

document.write("<p>Task 2 Output: Sum of even numbers = " + sum + "</p>");

// --- Task 3: Do-while loop to get user input until number > 10 ---

let userNum;

do {

userNum = Number(prompt("Enter a number greater than 10 (Task 3):"));

} while(userNum <= 10);

document.write("<p>Task 3 Output: You entered " + userNum + "</p>");

</script>

</body>

</html>

* **Functions**

(*Theory Assignment*)

Question 1: What are functions in JavaScript? Explain the syntax for

declaring and calling a function.

Ans:

* A **function** in JavaScript is a **block of reusable code** designed to perform a specific task. Functions help in **reducing code repetition**, **organizing code**, and making programs more **modular and maintainable**.
* Key points:

 Functions can **take inputs** called **parameters**.

 Functions can **return a value** using the return keyword.

 Functions must be **declared** before calling them (unless using function expressions with hoisting considerations).

* Syntax:

function functionName(parameters) {

// code to execute

return value; // optional

}

* **Calling a Function (Executing the Function)**
  + Calling (or invoking) a function executes the code within its body.
  + **Syntax:**

functionName(argument1, argument2, ...);

* + **Process:**
    - Use the **function's name**.
    - Follow it with **parentheses ()**.
    - Pass any required **arguments** (values) inside the parentheses.
* **Example using the declared functions above:**
  + // Calling the 'greet' function
    - let greetingMessage = greet("Charlie");
    - console.log(greetingMessage); // Output: "Hello, Charlie!"
  + // Calling the 'add' function
    - let sum = add(5, 3);
    - console.log(sum); // Output: 8
  + // Calling the 'isEven' function
    - let result = isEven(4);
    - console.log(result); // Output: true

The values passed ("Charlie", 5, 3, 4) are called **arguments**.

The variables used in the declaration (name, a, b, number) are called **parameters**.

Question 2: What is the difference between a function declaration and function expression?

Ans:

* Both **function declarations** and **function expressions** are ways to define functions in JavaScript, but they have **key differences** in syntax, hoisting, and usage.

**1. Function Declaration**

* Declares a function using the function keyword at the top level.
* **Hoisted**: Can be called **before its declaration** in the code.
* **Syntax:**

function greet(name) {

return "Hello, " + name + "!";

}

// Calling the function

console.log(greet("Alice")); // Hello, Alice!

**Key Points:**

* Hoisting allows calling the function before it is written in code.
* Ideal for functions that are used throughout the code.

**2. Function Expression**

* A function is defined and **assigned to a variable**.
* **Not hoisted**: Can only be called **after it is defined**.
* Can be **anonymous** (no name) or named.
* **Syntax:**

const greet = function(name) {

return "Hello, " + name + "!";

};

// Calling the function

console.log(greet("Bob")); // Hello, Bob!

**Key Points:**

* Useful for **passing functions as arguments** (callbacks).
* Can be anonymous, unlike function declarations.

***Key Differences:***

| **Feature** | **Function Declaration** | **Function Expression** |
| --- | --- | --- |
| **Hoisting** | Yes, can be called before declaration | No, must be defined first |
| **Syntax** | function name () {} | const name = function () {} |
| **Anonymous** | Cannot be anonymous | Can be anonymous |
| **Use Case** | General-purpose, reusable | Callback functions, dynamic assignment |

Question 3: Discuss the concept of parameters and return values in functions.

Ans:

* Functions in JavaScript often need to **accept input** and **provide output**. This is done using **parameters** and **return values**.

**1. Parameters**

* **Definition:** Parameters are **placeholders for input values** that you pass into a function.
* **Purpose:** Allow functions to **work with dynamic data** instead of fixed values.
* **Syntax:** Declared inside the parentheses in the function definition.
* **Example:**

function greet(name) {

console.log("Hello, " + name + "!");

}

// Calling the function with an argument

greet("Alice"); // Output: Hello, Alice!

greet("Bob"); // Output: Hello, Bob!

**Explanation:**

* name is the **parameter** of the function greet.
* "Alice" and "Bob" are **arguments** passed when calling the function.

**2. Return Values**

* **Definition:** A return value is the **output** a function produces and sends back to the code that called it.
* **Purpose:** Allows functions to **compute and provide results** instead of just performing actions.
* **Syntax:** Use the return keyword inside the function.
* **Example:**

function add(a, b) {

return a + b; // Returns the sum

}

let sum = add(5, 3); // sum stores the returned value

console.log(sum); // Output: 8

**Explanation:**

* a and b are **parameters**.
* add(5, 3) calls the function with **arguments** 5 and 3.
* The function **returns** 8, which is stored in the variable sum.

**Key Points:**

1. Parameters are **inputs**, return values are **outputs**.
2. A function can have **multiple parameters** or none at all.
3. A function can return a **value** using return or return **undefined** if no return is specified.

(*Lab Assignment*)

Task 1:

• Write a function greetUser that accepts a user’s name as a parameter and displays greeting message (e.g., "Hello, John!").

Task 2:

• Create a JavaScript function calculates Sum that takes two numbers as parameters, adds them, and returns the result.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>JavaScript Functions Tasks</title>

</head>

<body>

<h2>Task 1: Greet User</h2>

<h2>Task 2: Calculate Sum of Two Numbers</h2>

<script>

// --- Task 1: Function to greet user ---

function greetUser(name) {

document. write("<p>Hello, " + name + “! </p>");

}

// Example usage for Task 1

let userName = prompt ("Enter your name for Task 1:");

greetUser(userName);

// --- Task 2: Function to calculate sum ---

function calculateSum (num1, num2) {

let sum = num1 + num2;

document.write ("<p>Sum: " + sum + "</p>");

}

// Example usage for Task 2

let number1 = Number (prompt ("Enter first number for Task 2:"));

let number2 = Number (prompt ("Enter second number for Task 2:"));

calculateSum (number1, number2);

</script>

</body>

</html>

* **Arrays**

(*Theory Assignment*)

Question 1: What is an array in JavaScript? How do you declare and initialize an array?

Ans:

* An array in JavaScript is a special variable that can hold multiple values under a single name.
* Arrays are ordered collections of elements.
* Each element in an array has an index, starting from 0.
* Arrays can store any type of data, including numbers, strings, Booleans, objects, or even other arrays.
* **Declaring and Initializing Arrays:**

**1. Using Array Literal (Most Common)**

let fruits = ["Apple", "Banana", "Orange"];

* Here, fruits is an array containing three string elements.
* Indexes: fruits[0] = "Apple", fruits[1] = "Banana", fruits[2] = "Orange"

**2. Using Array Constructor**

let numbers = new Array(1, 2, 3, 4, 5);

* Creates an array with the specified elements.

**3. Empty Array and Adding Elements Later**

let colors = [];

colors[0] = "Red";

colors[1] = "Green";

colors[2] = "Blue";

* **Accessing Array Elements**

console.log(fruits[0]); // Output: Apple

console.log(numbers[2]); // Output: 3

* **Modifying Array Elements**

fruits[1] = "Mango";

console.log(fruits); // Output: ["Apple", "Mango", "Orange"]

* **Key Points:**

1. Arrays are **zero-indexed**.
2. They can contain **mixed data types**:

let mixed = [1, "Hello", true];

1. Arrays are **dynamic**, meaning you can **add or remove elements** at any time.

Question 2: Explain the methods push (), pop (), shift (), and unshift () used in arrays.

Ans:

* JavaScript provides several **built-in methods** to manipulate arrays. Four commonly used methods are: **push(), pop(), shift(), and unshift()**.

**1. push()**

* **Purpose:** Adds **one or more elements to the end** of an array.
* **Returns:** The **new length** of the array.
* **Example:**

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

fruits.push ("Orange");

console.log(fruits); // Output: ["Apple", "Banana", "Orange"]

**2. pop()**

* **Purpose:** Removes the **last element** from an array.
* **Returns:** The **removed element**.
* **Example:**

let fruits = ["Apple", "Banana", "Orange"];

let removed = fruits. pop();

console.log(fruits); // Output: ["Apple", "Banana"]

console.log(removed); // Output: "Orange"

**3. shift()**

* **Purpose:** Removes the **first element** of an array.
* **Returns:** The **removed element**.
* **Example:**

let fruits = ["Apple", "Banana", "Orange"];

let removed = fruits.shift();

console.log(fruits); // Output: ["Banana", "Orange"]

console.log(removed); // Output: "Apple"

**4. unshift()**

* **Purpose:** Adds **one or more elements to the beginning** of an array.
* **Returns:** The **new length** of the array.
* **Example:**

let fruits = ["Banana", "Orange"];

fruits.unshift("Apple");

console.log(fruits); // Output: ["Apple", "Banana", "Orange"]

* **Summary of Methods**

| **Method** | **Action** | **Returns** |
| --- | --- | --- |
| push() | Add elements at the **end** | New array length |
| pop() | Remove element from the **end** | Removed element |
| shift() | Remove element from the **start** | Removed element |
| unshift() | Add elements at the **start** | New array length |

* **Key Points:**
* push() and unshift() **add elements**.
* pop() and shift() **remove elements**.
* These methods **modify the original array**.

(*Lab Assignment*)

Task 1:

• Declare an array of fruits (["apple", "banana", "cherry"]).

Use JavaScript to:

• Add a fruit to the end of the array.

• Remove the first fruit from the array.

• Log the modified array to the console.

Task 2:

• Write a program to find the sum of all elements in an array

of numbers.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>JavaScript Array Tasks</title>

</head>

<body>

<h2>Task 1: Modify Fruits Array</h2>

<script>

// --- Task 1: Modify fruits array ---

let fruits = ["apple", "banana", "cherry"];

document. write ("<p>Original array: " + fruits + "</p>");

// Add a fruit to the end

fruits.push("orange");

document.write("<p>After push(): " + fruits + "</p>");

// Remove the first fruit

fruits.shift();

document.write("<p>After shift(): " + fruits + "</p>");

// Final modified array

document. Write ("<p>Modified array: " + fruits + "</p>");

</script>

<h2>Task 2: Sum of Array Elements</h2>

<script>

// --- Task 2: Sum of all elements in a numbers array ---

let numbers = [5, 10, 15, 20];

let sum = 0;

for (let i = 0; i < numbers.length; i++) {

sum += numbers[i];

}

document. write("<p>Numbers array: " + numbers + "</p>");

document. write("<p>Sum of elements: " + sum + "</p>");

</script>

</body>

</html>

* **Objects**

(*Theory Assignment*)

Question 1: What is an object in JavaScript? How are objects different from arrays?

Ans:

* An object in JavaScript is a collection of properties, where each property is a key-value pair.
* The key (also called property name) is always a string (or symbol).
* The value can be any data type, including numbers, strings, arrays, functions, or even other objects.
* Objects are used to store and organize data in a structured way.
* Syntax:

let person = {

name: "John",

age: 25,

profession: "Developer"

};

* **Accessing Object Properties:**

console.log(person.name); // Output: John

console.log(person["age"]); // Output: 25

* **Modifying Object Properties:**

person.age = 26;

person.country = "USA"; // Adding a new property

console.log(person);

* **How the Objects are Different from Array:**
* Object v/s Array:

| **Feature** | **Object** | **Array** |
| --- | --- | --- |
| **Structure** | Key-value pairs | Ordered list of elements |
| **Indexing** | Accessed by key (string) | Accessed by index (number) |
| **Use Case** | Represent complex data, entities | Store lists or collections |
| **Example** | {name: "John", age: 25} | ["apple", "banana", "cherry"] |

* + **Example of Difference:**

let fruits = ["apple", "banana", "cherry"];

console.log(fruits[1]); // Output: banana (accessed by index)

let person = {name: "Alice", age: 30};

console.log(person.name); // Output: Alice (accessed by key)

Question 2: Explain how to access and update object properties using dot notation and bracket notation.

Ans:

* In JavaScript, **object properties** can be accessed and modified using two main methods: **dot notation** and **bracket notation**.

**1. Dot Notation**

* Use a **dot (.)** followed by the property name.
* Works only when the property name is a **valid identifier** (no spaces, special characters, or numbers at the start).
* **Example:**

let person = {

name: "Alice",

age: 25

};

// Access properties

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

console.log(person.age); // Output: 25

// Update properties

person.age = 26;

console.log(person.age); // Output: 26

**2. Bracket Notation**

* Use **square brackets [ ]** with the property name as a **string**.
* Useful when the property name contains **spaces, special characters, or is dynamic**.
* **Example:**

let person = {

"full name": "Alice Johnson",

age: 25

};

// Access properties

console.log(person["full name"]); // Output: Alice Johnson

// Update properties

person["full name"] = "Alice Smith";

console.log(person["full name"]); // Output: Alice Smith

// Dynamic property access

let prop = "age";

console.log(person[prop]); // Output: 25

* **Key Differences Between Dot and Bracket Notation:**

| **Feature** | **Dot Notation** | **Bracket Notation** |
| --- | --- | --- |
| Syntax | object.property | object["property"] |
| Property Names | Must be valid identifiers | Can be any string, including spaces or special characters |
| Dynamic Access | Not possible | Possible (e.g., object[var]) |

(*Lab Assignment*)

Task:

• Create a JavaScript object car with properties brand, model, and year. Use JavaScript to:

• Access and print the car’s brand and model.

• Update the year property.

• Add a new property color to the car object.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>JavaScript Object Task</title>

</head>

<body>

<h2>Car Object Example</h2>

<script>

// Create car object

let car = {

brand: "Toyota",

model: "Corolla",

year: 2020

};

// Access and print brand and model

document.write("<p>Brand: " + car.brand + "</p>");

document.write("<p>Model: " + car.model + "</p>");

// Update the year property

car.year = 2023;

document.write("<p>Updated Year: " + car.year + "</p>");

// Add a new property 'color'

car.color = "Red";

document.write("<p>Color: " + car.color + "</p>");

// Display the complete object

document.write("<p>Complete Car Object: " + JSON.stringify(car) + "</p>");

</script>

</body>

</html>

* **JavaScript Events**

(*Theory Assignment*)

Question 1: What are JavaScript events? Explain the role of event listeners.

Ans:

* **Events** in JavaScript are **actions or occurrences** that happen in the browser, which the code can respond to.
* Events can be triggered by the **user**, like clicking a button, typing in an input field, moving the mouse, or submitting a form.
* They can also be triggered **programmatically** by the browser or JavaScript code itself.

**Examples of common events:**

* click – When the user clicks an element.
* mouseover – When the mouse pointer moves over an element.
* keydown – When a key is pressed.
* submit – When a form is submitted.
* Role of Event Listeners:
  + An **event listener** is a function in JavaScript that **waits for a specific event to occur** on a particular element and then executes code in response.

**Advantages of using event listeners:**

1. Allows separation of HTML and JavaScript code.
2. Multiple listeners can be added to a single element.
3. Offers better control over event handling compared to inline events.

* **Syntax of an Event Listener**

element.addEventListener("event", function() {

// code to execute when event occurs

});

* **Example:**

let button = document.getElementById("myButton");

// Add click event listener

button.addEventListener("click", function() {

alert("Button clicked!");

});

* **Explanation:**
* myButton is the HTML element.
* "click" is the event type.
* The function executes when the event occurs.

Question 2: How does the addEventListener () method work in JavaScript? Provide an example.

Ans:

* The addEventListener() method is used to attach an event handler to an HTML element.
  + It allows your code to respond to specific events like clicks, mouse movements, key presses, and more.
  + Multiple event listeners can be attached to a single element without overwriting each other.
  + Using addEventListener() is considered a best practice over inline event attributes like onclick.
  + Syntax:
    - element.addEventListener(event, function, useCapture);

**Parameters:**

* + - * + event – A string representing the event type (e.g., "click", "mouseover", "keydown").
        + function – The function to execute when the event occurs.
        + useCapture *(optional)* – Boolean indicating whether the event should be captured in the capturing phase (true) or bubbling phase (false). Default is false.
* Example:

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

<script>

// Select the button element

let button = document.getElementById("myButton");

// Add a click event listener

button.addEventListener("click", function() {

alert("Button clicked!");

});

</script>

**Explanation:**

* getElementById("myButton") selects the button element.
* addEventListener("click", function(){...}) attaches a function that executes when the button is clicked.
* You can attach multiple event listeners to the same element without overwriting previous ones.
* **Advantages of addEventListener():**
  + Keeps JavaScript separate from HTML.
  + Supports multiple event handlers on the same element.
  + Provides better control over event propagation (capturing and bubbling).

(*Lab Assignment*)

Task:

• Create a simple webpage with a button that, when clicked, displays an alert

Saying "Button clicked!" using JavaScript event listeners.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Button Click Event</title>

<style>

button {

padding: 10px 20px;

font-size: 16px;

cursor: pointer;

}

</style>

</head>

<body>

<h2>Click the Button</h2>

<!-- Using onclick attribute -->

<button onclick="buttonClicked()">Click Me!</button>

<script>

// Function to execute on click

function buttonClicked() {

alert("Button clicked!");

}

</script>

</body>

</html>

* **DOM Manipulation**

(*Theory Assignment*)

Question 1: What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?

Ans:

* The **DOM**, or **Document Object Model**, is a **programming interface** for HTML and XML documents.
  + It represents the page so that **programs can change the document structure, style, and content dynamically**.
  + In the DOM, **every element, attribute, and text is represented as an object**, which can be manipulated using JavaScript.
* **Key Points:**

1. The DOM **creates a tree-like structure** of the HTML document.
   * Example: The <body> contains <h1>, <p>, <button>, etc., as child nodes.
2. JavaScript can **access, modify, add, or remove elements** in this tree.
3. The DOM provides **methods and properties** to interact with HTML elements.

* **How JavaScript Interacts with the DOM**

JavaScript interacts with the DOM using **DOM methods**. Common actions include:

1. **Accessing elements**

let heading = document.getElementById("myHeading");

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

let buttons = document.querySelectorAll("button");

1. **Changing content**

heading.innerText = "New Heading";

1. **Changing styles**

heading.style.color = "blue";

heading.style.fontSize = "24px";

1. **Adding or removing elements**

let newPara = document.createElement("p");

newPara.innerText = "This is a new paragraph.";

document.body.appendChild(newPara); // Adds to the body

1. **Handling events**

let button = document.getElementById("myButton");

button.addEventListener("click", function() {

alert("Button clicked!");

});

Question 2: Explain the methods getElementById (),

getElementsByClassName () and querySelector () used to

select elements from the DOM.

Ans:

* JavaScript provides several methods to **select elements from the DOM** so you can manipulate them. Three commonly used methods are:

**1. getElementById()**

* **Purpose:** Selects a **single element** by its id attribute.
* **Returns:** The **element object** if found, otherwise null.
* **Usage:** Ideal for selecting **unique elements**.
* **Example:**

<h1 id="myHeading">Hello World</h1>

<script>

let heading = document.getElementById("myHeading");

heading.style.color = "blue"; // Changes text color to blue

</script>

**2. getElementsByClassName()**

* **Purpose:** Selects **all elements** that have a specific class name.
* **Returns:** An **HTMLCollection** (similar to an array, but not exactly).
* **Usage:** Useful for selecting multiple elements sharing the same class.
* **Example:**

<p class="para">Paragraph 1</p>

<p class="para">Paragraph 2</p>

<script>

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

for (let i = 0; i < paragraphs.length; i++) {

paragraphs[i].style.fontWeight = "bold"; // Makes text bold

}

</script>

**3. querySelector()**

* **Purpose:** Selects the **first element** that matches a CSS selector.
* **Returns:** The element object if found, otherwise null.
* **Usage:** Can select elements by **id, class, tag, or complex CSS selectors**.
* **Example:**

<p class="para">First paragraph</p>

<script>

let firstPara = document.querySelector(".para");

firstPara.style.color = "red"; // Changes text color to red

</script>

* **Key Points:**
* Use getElementById() for **unique IDs**.
* Use getElementsByClassName() for **multiple elements with the same class**.
* Use querySelector() for **flexible CSS-style selection**.
* **Summary Table:**

| **Method** | **Returns** | **Selects** | **Example** |
| --- | --- | --- | --- |
| getElementById("id") | Single element | Element by id | #myHeading |
| getElementsByClassName() | HTMLCollection | Elements by class | .para |
| querySelector() | Single element | First element matching CSS selector | .para or #myHeading |

(*Lab Assignment*)

Task:

• Create an HTML page with a paragraph (<p>) that displays "Hello, World!".

• Use JavaScript to:

• Change the text inside the paragraph to "JavaScript is

fun!".

• Change the color of the paragraph to blue.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>DOM Manipulation Task</title>

</head>

<body>

<h2>DOM Example</h2>

<p id="myPara" style="color:red;">Hello, World!</p> <!-- Original text -->

<script>

// Select the paragraph by ID

let para = document.getElementById("myPara");

// Change the text

para.innerText = "JavaScript is fun!";

// Change the color

para.style.color = "blue";

</script>

</body>

</html>

* **JavaScript Timing Events (setTimeout, setInterval)**

(*Theory Assignment*)

Question 1: Explain the setTimeout () and setInterval () functions in JavaScript. How they used for timing events?

Ans:

**1) setTimeout() Function**

* **Definition:** Executes a function **once** after a specified amount of time (in milliseconds).
* **Syntax:**
* setTimeout(function, delay);
* **Use case:** Useful when you want to **delay an action** (e.g., showing a message after 3 seconds).

**Example:**

setTimeout(function() {

alert("Hello after 3 seconds!");

}, 3000);

Here, the alert will appear **once** after 3 seconds.

**2) setInterval() Function**

* **Definition:** Executes a function **repeatedly** at a specified time interval (in milliseconds).
* **Syntax:**
* setInterval(function, interval);
* **Use case:** Useful when you want to **repeat an action** (e.g., updating a clock every second).

**Example:**

setInterval(function() {

console.log("This prints every 2 seconds");

}, 2000);

This will keep printing the message **every 2 seconds** until stopped.

* **Stopping Timers**

Both functions return an **ID** that can be used to stop them:

* clearTimeout(id) → Stops a timeout before it runs.
* clearInterval(id) → Stops a repeated interval.

**Example:**

let timer = setInterval(() => console.log("Running..."), 1000);

// Stop after 5 seconds

setTimeout(() => clearInterval(timer), 5000);

* **Summary**
* **setTimeout()** → Runs **once** after a delay.
* **setInterval()** → Runs **repeatedly** at regular intervals.
* Both can be stopped using clearTimeout() or clearInterval().

Question 2: Provide an example of how to use setTimeout () to delay an action by 2 seconds.

Ans:

* Here’s a **simple example** showing how to use setTimeout () to delay an action by **2 seconds (2000 milliseconds):**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>setTimeout Example</title>

</head>

<body>

<h2>setTimeout() Example</h2>

<p id="message">Wait for 2 seconds...</p>

<script>

// Use setTimeout to delay the action

setTimeout(function() {

document.getElementById("message").innerText = "Action executed after 2 seconds!";

}, 2000);

</script>

</body>

</html>

(*Lab Assignment*)

Task 1:

• Write a program that changes the background color of a webpage

after 5 seconds using setTimeout ().

Task 2:

• Create a digital clock that updates every second using setInterval ().

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>setTimeout & setInterval Tasks</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

margin-top: 100px;

}

#clock {

font-size: 2em;

font-weight: bold;

color: darkblue;

}

</style>

</head>

<body>

<h2>Task 1: Change Background Color After 5 Seconds</h2>

<p>Wait for 5 seconds and see the background color change!</p>

<h2>Task 2: Digital Clock</h2>

<p id="clock"></p>

<script>

// --- Task 1: Change background color after 5 seconds ---

setTimeout(function() {

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

}, 5000);

// --- Task 2: Digital Clock using setInterval ---

function updateClock() {

let now = new Date();

let time = now.toLocaleTimeString();

document.getElementById("clock").innerText = time;

}

// Call updateClock every 1 second

setInterval(updateClock, 1000);

// Call once immediately so it shows without waiting

updateClock();

</script>

</body>

</html>