**Javascript Essentials And Advanced**

**JavaScript Introduction**

**Theory Assignment**

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

**JavaScript** is a high-level, interpreted programming language used to create dynamic and interactive behavior on websites. It allows developers to manipulate the DOM, respond to user events, perform form validation, send/receive data from servers (using APIs), and much more.

**Role in Web Development:**

* Makes web pages interactive (e.g., image sliders, modals).
* Allows form validations without page reloads.
* Enables AJAX for asynchronous data loading.
* Powers modern front-end frameworks like React, Angular, Vue.

**Question 2: How is JavaScript different from other programming languages like Python or Java?**

| **Feature** | **JavaScript** | **Python** | **Java** |
| --- | --- | --- | --- |
| Runtime | Browser (client-side), Node.js | Server-side | Server-side (JVM) |
| Typing | Dynamically typed | Dynamically typed | Statically typed |
| Syntax | C-style syntax | English-like, indentation-based | Strict and verbose |
| Use Case | Web interactivity, front-end | Scripting, data science, back-end | Enterprise applications |
| Compilation | Interpreted | Interpreted | Compiled to bytecode (JVM) |

**Question 3: Discuss the use of <script> tag in HTML. How can you link an external JavaScript file to an HTML document?**

The <script> tag is used to embed or reference JavaScript in an HTML page.

**➤ To include inline JavaScript:**

<script>

alert("Hello from inline JavaScript!");

</script>

**➤ To link external JavaScript file:**

<script src="script.js"></script>

The src attribute points to the location of the external JavaScript file.

**Lab Assignment**

**Task: Create a simple HTML page and add a <script> tag**

**HTML + JavaScript Example:**

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>JavaScript Introduction</title>

</head>

<body>

<h1>Welcome to My Page</h1>

<!-- Inline JavaScript -->

<script>

alert("Welcome to JavaScript!");

</script>

</body>

</html>

**OR Using External Script**

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>External JS Example</title>

</head>

<body>

<h1>Hello, World!</h1>

<script src="script.js"></script>

</body>

</html>

**script.js**

alert("Welcome to JavaScript!");

## Variables and Data Types

### Theory Assignment

### ****Question 1: What are variables in JavaScript? How do you declare a variable using var, let, and const?****

* **Variables** are containers used to store data values.
* In JavaScript, variables can be declared using three keywords:
  1. **var**: Function-scoped, can be re-declared and updated.
  2. **let**: Block-scoped, can be updated but not re-declared in the same scope.
  3. **const**: Block-scoped, cannot be updated or re-declared (used for constants).

var name = "Alice";

let age = 25;

const PI = 3.14;

### ****Question 2: Explain the different data types in JavaScript. Provide examples for each.****

* **String:** Represents text.
* let name = "John";
* **Number:** Represents both integer and floating-point numbers.
* let age = 30;
* let price = 99.99;
* **Boolean:** Represents true or false.
* let isStudent = true;
* **Null:** Represents an intentional absence of any object value.
* let emptyValue = null;
* **Undefined:** Variable declared but not assigned a value.
* let x;
* **Object:** Collection of key-value pairs.
* let person = { name: "Alice", age: 25 };
* **Symbol:** Unique and immutable primitive value (advanced use).

### ****Question 3: What is the difference between undefined and null in JavaScript?****

* **undefined** means a variable has been declared but not assigned any value.
* **null** is an assignment value that represents "no value" or "empty".

let a; // undefined

let b = null; // null

### Lab Assignment (Variables and Data Types)

// Declare variables of different data types

let name = "Alice"; // string

let age = 30; // number

let isStudent = true; // boolean

let emptyValue = null; // null

let notAssigned; // undefined

// Log values and their types

console.log(name, typeof name);

console.log(age, typeof age);

console.log(isStudent, typeof isStudent);

console.log(emptyValue, typeof emptyValue);

console.log(notAssigned, typeof notAssigned);

## JavaScript Operators

### Theory Assignment

### ****Question 1: What are the different types of operators in JavaScript? Explain with examples.****

* **Arithmetic Operators:** Used for mathematical operations.
* +, -, \*, /, %
* **Assignment Operators:** Assign values to variables.
* =, +=, -=, \*=, /=
* **Comparison Operators:** Compare two values and return a boolean.
* ==, ===, !=, !==, >, <, >=, <=
* **Logical Operators:** Used with boolean values.
* && (AND), || (OR), ! (NOT)

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

* == checks for equality of values after type coercion (loose equality).
* === checks for equality of both value and type (strict equality).

5 == "5"; // true

5 === "5"; // false

### Lab Assignment (Operators)

let a = 20;

let b = 5;

// Arithmetic operations

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

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

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

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

// Comparison

console.log("Are a and b equal (==)?", a == b);

console.log("Is a greater than b?", a > b);

// Logical operators

console.log("Is a > 10 AND b < 5?", (a > 10) && (b < 5));

console.log("Is a > 10 OR b < 5?", (a > 10) || (b < 5));

## Control Flow (If-Else, Switch)

### Theory Assignment

### ****Question 1: What is control flow in JavaScript? Explain how if-else statements work with an example.****

* **Control flow** in JavaScript determines the order in which statements or blocks of code are executed based on conditions.
* **if-else statements** allow the program to execute certain code if a condition is true and another code block if it is false.

**Example:**

let num = 10;

if (num > 0) {

console.log("Positive number");

} else if (num < 0) {

console.log("Negative number");

} else {

console.log("Zero");

}

### ****Question 2: Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?****

* A **switch statement** tests a variable or expression against multiple cases and executes the matching case block.
* Use a switch statement when you have many conditions based on the same variable, making code cleaner and more readable than multiple if-else statements.

**Example:**

let day = 3;

switch(day) {

case 1:

console.log("Monday");

break;

case 2:

console.log("Tuesday");

break;

case 3:

console.log("Wednesday");

break;

default:

console.log("Invalid day");

}

### Lab Assignment - Control Flow

#### Task 1: Check if a number is positive, negative, or zero using if-else

let number = -5;

if (number > 0) {

console.log("Positive number");

} else if (number < 0) {

console.log("Negative number");

} else {

console.log("Zero");

}

#### Task 2: Use switch to display day of the week based on user input

let day = 4; // Example input

switch(day) {

case 1:

console.log("Monday");

break;

case 2:

console.log("Tuesday");

break;

case 3:

console.log("Wednesday");

break;

case 4:

console.log("Thursday");

break;

case 5:

console.log("Friday");

break;

case 6:

console.log("Saturday");

break;

case 7:

console.log("Sunday");

break;

default:

console.log("Invalid day");

}

## 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.****

* **for loop:** Runs a block of code a specific number of times.

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

console.log(i);

}

* **while loop:** Runs a block of code while a condition is true.

let i = 1;

while(i <= 5) {

console.log(i);

i++;

}

* **do-while loop:** Runs the block of code once before checking the condition, then repeats as long as the condition is true.

let i = 1;

do {

console.log(i);

i++;

} while(i <= 5);

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

* A **while loop** checks the condition before executing the code block.
* A **do-while loop** executes the code block at least once before checking the condition.

### Lab Assignment - Loops

#### Task 1: Print numbers 1 to 10 using a for loop

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

console.log(i);

}

#### Task 2: Sum all even numbers between 1 and 20 using a while loop

let i = 1;

let sum = 0;

while(i <= 20) {

if(i % 2 === 0) {

sum += i;

}

i++;

}

console.log("Sum of even numbers between 1 and 20:", sum);

#### Task 3: Use a do-while loop to prompt user input until a number greater than 10 is entered

let num;

do {

num = parseInt(prompt("Enter a number greater than 10:"));

} while(num <= 10);

console.log("You entered:", num);

## Functions

### Theory Assignment

### ****Question 1: What are functions in JavaScript? Explain the syntax for declaring and calling a function.****

* Functions are reusable blocks of code designed to perform a particular task.
* You **declare** a function using the function keyword, followed by the function name and parentheses ().
* You **call** a function by writing its name followed by parentheses.

**Syntax:**

function functionName(parameters) {

// code to be executed

}

// Calling the function

functionName(arguments);

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

* **Function Declaration:** Named function defined with the function keyword; it gets hoisted, so it can be called before declaration.

function greet() {

console.log("Hello!");

}

* **Function Expression:** Function assigned to a variable; not hoisted, so it cannot be called before assignment.

const greet = function() {

console.log("Hello!");

};

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

* **Parameters** are placeholders for values that a function accepts when called.
* **Return value** is the output a function gives back after execution using the return statement.

Example:

function add(a, b) {

return a + b;

}

let result = add(2, 3); // result is 5

### Lab Assignment - Functions

#### Task 1: Function greetUser to display greeting message

function greetUser(name) {

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

}

greetUser("John"); // Output: Hello, John!

#### Task 2: Function calculateSum to add two numbers and return result

function calculateSum(num1, num2) {

return num1 + num2;

}

let sum = calculateSum(5, 7);

console.log("Sum:", sum); // Output: Sum: 12

## Arrays

### Theory Assignment

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

* An **array** is a data structure that stores multiple values in a single variable.
* You declare an array using square brackets [] and initialize it by listing values separated by commas.

Example:

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

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

* push(): Adds one or more elements to the **end** of an array.

fruits.push("orange");

* pop(): Removes the last element from an array.

fruits.pop();

* shift(): Removes the first element from an array.

fruits.shift();

* unshift(): Adds one or more elements to the **beginning** of an array.

fruits.unshift("mango");

### Lab Assignment - Arrays

#### Task 1: Work with fruits array

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

// Add a fruit to the end

fruits.push("orange");

// Remove the first fruit

fruits.shift();

// Log the modified array

console.log(fruits); // Output: ["banana", "cherry", "orange"]

#### Task 2: Sum all elements in an array of numbers

let numbers = [1, 2, 3, 4, 5];

let total = 0;

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

total += numbers[i];

}

console.log("Sum of numbers:", total); // Output: Sum of numbers: 15

## Objects

### Theory Assignment

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

* An **object** is a collection of key-value pairs (properties), where each key (property name) is a string (or symbol), and the value can be any data type (including functions).
* Objects are used to represent complex entities with multiple attributes.
* **Difference from arrays:**
  + Arrays are ordered collections accessed by numeric indices.
  + Objects store data as named properties and are **unordered**.

Example of an object:

let car = {

brand: "Toyota",

model: "Corolla",

year: 2020

};

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

* **Dot notation:** Access or update properties using a dot . followed by the property name.

console.log(car.brand); // Access

car.year = 2021; // Update

* **Bracket notation:** Access or update properties using square brackets with the property name as a string.

console.log(car["model"]); // Access

car["color"] = "red"; // Add new property

### Lab Assignment — Objects

// Create the car object

let car = {

brand: "Toyota",

model: "Corolla",

year: 2020

};

// Access and print brand and model

console.log("Brand:", car.brand);

console.log("Model:", car.model);

// Update the year property

car.year = 2022;

// Add a new property color

car.color = "Red";

// Log the updated object

console.log(car);

**Output:**

Brand: Toyota

Model: Corolla

{ brand: 'Toyota', model: 'Corolla', year: 2022, color: 'Red' }

## JavaScript Events

### Theory Assignment

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

* **Events** are actions or occurrences that happen in the browser, like clicks, key presses, mouse movements, or page load.
* **Event listeners** are functions that wait for an event to occur on a specific element and then run some code in response.

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

* addEventListener() attaches an event handler function to an element for a specific event.
* Syntax:

element.addEventListener(event, function);

Example:

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

alert("Button clicked!");

});

### Lab Assignment — JavaScript Events

Here’s a simple HTML page with a button and JavaScript to alert when clicked:

<!DOCTYPE html>

<html>

<head>

<title>Button Click Event</title>

</head>

<body>

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

<script>

// Select the button by its ID

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

// Add click event listener

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

alert("Button clicked!");

});

</script>

</body>

</html>

## DOM Manipulation

### Theory Assignment

#### Question 1: What is the DOM (Document Object Model) in JavaScript?

* The **DOM** is a programming interface for HTML and XML documents.
* It represents the page as a **tree of nodes**, where each node is an object representing part of the page.
* JavaScript uses the DOM to **access, modify, add, or delete** elements and content on a web page dynamically.

#### Question 2: Explain the methods:

* **getElementById()**: Selects an element by its unique id.
* **getElementsByClassName()**: Returns a list of elements with the specified class.
* **querySelector()**: Returns the **first** element that matches a CSS selector.

document.getElementById("demo");

document.getElementsByClassName("box")[0];

document.querySelector("p"); // or ".class", "#id"

### Lab Assignment

**HTML + JS: Change paragraph text and color**

<!DOCTYPE html>

<html>

<head>

<title>DOM Manipulation</title>

</head>

<body>

<p id="myPara">Hello, World!</p>

<script>

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

para.textContent = "JavaScript is fun!";

para.style.color = "blue";

</script>

</body>

</html>

## JavaScript Timing Events

### Theory Assignment

#### Question 1: What is setTimeout() and setInterval()?

* **setTimeout()**: Runs a function once **after a specified delay** (in milliseconds).
* **setInterval()**: Runs a function **repeatedly** at every specified interval.

#### Question 2: Example of setTimeout() with 2 seconds delay:

setTimeout(function() {

console.log("This message is shown after 2 seconds");

}, 2000);

### Lab Assignment

#### Task 1: Change background color after 5 seconds

<!DOCTYPE html>

<html>

<head><title>Background Change</title></head>

<body>

<script>

setTimeout(function() {

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

}, 5000);

</script>

</body>

</html>

#### Task 2: Digital Clock using setInterval()

<!DOCTYPE html>

<html>

<head><title>Digital Clock</title></head>

<body>

<h1 id="clock"></h1>

<script>

function updateClock() {

const now = new Date();

const time = now.toLocaleTimeString();

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

}

updateClock(); // initial call

setInterval(updateClock, 1000); // update every 1 second

</script>

</body>

</html>

## JavaScript Error Handling

### Theory Assignment

#### Question 1: What is error handling in JavaScript?

* **Error handling** is the process of catching and managing errors in a program.
* JavaScript uses:
  + try: Block of code to test for errors.
  + catch: Block to handle the error.
  + finally: (optional) Runs after try/catch regardless of result.

try {

// risky code

} catch (error) {

// handle error

} finally {

// always runs

}

#### Question 2: Why is error handling important?

* Prevents the script from crashing.
* Provides user-friendly messages.
* Helps in debugging.

### Lab Assignment: Handle divide-by-zero error

<!DOCTYPE html>

<html>

<head><title>Error Handling</title></head>

<body>

<script>

try {

let a = 10;

let b = 0;

if (b === 0) {

throw new Error("Cannot divide by zero");

}

let result = a / b;

console.log(result);

} catch (err) {

console.error("Error:", err.message);

} finally {

console.log("Execution complete.");

}

</script>

</body>

</html>