JavaScript

~JAVASCRIPT INTRODUCTION

❖What is JavaScript? Explain the role of JavaScript in web development.

ANSWER: JavaScript is a programming language used to make web pages interactive and dynamic.

- **HTML → structure**
- **CSS → design**
- **JavaScript → behavior**

Role in web development:

- **Adds interactivity (buttons, forms, animations)**
- **Updates content dynamically without reloading**
- **Handles user actions (clicks, inputs)**
- **Enables modern web apps (React, Angular, etc.)**

❖ How is JavaScript different from other programming languages like Python or Java?

ANSWER: JavaScript is mainly used for making web pages interactive and runs directly in browsers, while Python and Java are general-purpose languages that need separate interpreters. JavaScript has a simpler, web-focused design; Python is easy to read and used for data and automation; and Java is more structured, used for large applications and Android development.

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

ANSWER: The <script> tag is used in HTML to add JavaScript. It can contain code directly or link an external file using the src attribute, like <script src="script.js"></script>, which keeps the code organized and easy to manage.

~Variables and Data Types

❖What are variables in JavaScript? How do you declare a variable using var, let, and const?

ANSWER: Variables in JavaScript are used to store data values. They can be declared using var, let, or const. The var keyword is function-scoped and can be redeclared, let is block-scoped and can be updated but not redeclared, and const is block-scoped and cannot be changed once assigned. In modern JavaScript, let and const are preferred for better code control.

❖Explain the different data types in JavaScript. Provide examples for each.

ANSWER: JavaScript has several data types that are mainly divided into primitive and non-primitive (object) types.

1. Primitive Data Types:

- **String – Used for text.**

  let name = "Sarthak";

- **Number – Represents numeric values.**

  let age = 25;

- **Boolean – Holds true or false.**

  let isStudent = true;

- **Undefined – A variable declared but not assigned a value.**

  let x;

- **Null – Represents an empty or unknown value.**

  let y = null;

- **Symbol – Used for unique identifiers.**

  let id = Symbol("id");

- **BigInt – For very large integers.**

  let bigNum = 1234567890123456789Ɂn;

2. Non-Primitive Data Type:

- **Object – Used to store collections of data.**

  let person = { name: "Sarthak", age: 25 };

❖What is the difference between undefined and null in JavaScript?

ANSWER: undefined is the default for unassigned variables, while null is intentionally used to show an empty or non-existent value.

~JavaScript Operators

❖What are the different types of operators in JavaScript? Explain with examples. Arithmetic operators , Assignment operators , Comparison operators , Logical operators.

ANSWER: JavaScript has several types of operators used to perform different actions on values and variables.

1. **Arithmetic Operators – Used for mathematical calculations.**

```
let a = 10, b = 5;
console.log(a + b); // Addition → 15
console.log(a - b); // Subtraction → 5
console.log(a * b); // Multiplication → 50
console.log(a / b); // Division → 2
console.log(a % b); // Modulus → 0
console.log(a ** 2); // Exponentiation → 100
```

2. Assignment Operators – Used to assign or modify variable values.

```
let x = 10;
x += 5; // x = x + 5 → 15
x -= 2; // x = x - 2 → 13
x *= 2; // x = x * 2 → 26
x /= 2; // x = x / 2 → 13
```

3. Comparison Operators – Used to compare values (return true/false).

```
let a = 10, b = 5;
console.log(a > b);  // true
console.log(a < b);  // false
console.log(a == 10); // true
console.log(a === "10"); // false (strict equality)
console.log(a != b);  // true
```

4. Logical Operators – Used to combine conditions.

```
let x = true, y = false;
```

```
console.log(x && y); // AND → false
console.log(x || y); // OR → true
console.log(!x);     // NOT → false
```

❖ What is the difference between == and === in JavaScript?

ANSWER: == checks *value equality* with type conversion, while === checks *value and type equality* without conversion.

~Control Flow (If-Else, Switch)

❖ What is control flow in JavaScript? Explain how if-else statements work with an example.

ANSWER: Control flow in JavaScript refers to the order in which code is executed in a program. By default, JavaScript runs code from top to bottom, but control flow statements like if-else, loops, and switch can change this order based on conditions.

if-else Statement:

The if-else statement is used to run different blocks of code depending on whether a condition is true or false.

Syntax:

```
if (condition) {
  // code runs if condition is true
} else {
  // code runs if condition is false
}
```

Example:

```
let marks = 75;

if (marks >= 50) {
  console.log("Pass");
} else {
  console.log("Fail");
}
```

❖ Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?

ANSWER: A switch statement in JavaScript is used to execute different code blocks based on the value of a variable or expression. It's a cleaner and more organized alternative to

using multiple if-else statements when you need to compare the same variable with many possible values.

Syntax:

```
switch (expression) {
 case value1:
  // code if expression === value1
  break;
 case value2:
  // code if expression === value2
  break;
 default:
  // code if no case matches
}
```

~Loops (For, While, Do-While)

- ❖ **Explain the different types of loops in JavaScript (for, while, do-while). Provide a basic example of each.**

ANSWER: JavaScript provides several types of loops to execute a block of code repeatedly until a certain condition is met.

1. for Loop

Used when you know how many times you want to run the loop.

```
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
// Output: 1 2 3 4 5
```

2. while Loop

Runs as long as the condition is true. The condition is checked before each iteration.

```
let i = 1;
while (i <= 5) {
  console.log(i);
  i++;
}
// Output: 1 2 3 4 5
```

3. do-while Loop

Runs the code at least once, then repeats while the condition is true (condition checked after execution).

```
let i = 1;

do {

  console.log(i);

  i++;

} while (i <= 5);

// Output: 1 2 3 4 5
```

❖What is the difference between a while loop and a do-while loop?

ANSWER: while loop: The condition is checked before the code runs. If the condition is false at the start, the loop may not run at all.

~do-while loop: The code runs at least once, and the condition is checked after execution.

~Functions

❖What are functions in JavaScript? Explain the syntax for declaring and calling a function.

ANSWER: Functions in JavaScript are reusable blocks of code designed to perform a specific task. They help make code modular, reusable, and easier to manage.

Syntax for Declaring a Function:

```
function functionName(parameters) {
  // code to execute
}
```

Syntax for Calling a Function:

```
functionName(arguments);
```

❖What is the difference between a function declaration and a function expression?

ANSWER: 1. Function Declaration

- Defined using the function keyword with a name.
- Hoisted — can be called before it is defined in the code.

2. Function Expression

- A function assigned to a variable (can be anonymous or named).
- Not hoisted — can be called only after it is defined.

❖Discuss the concept of parameters and return values in functions.

ANSWER: Parameters are variables listed inside the parentheses of a function definition. They act as placeholders for values (called arguments) that are passed when the function is called.

Return values are the results that a function sends back to the code that called it using the return statement. Once a value is returned, the function stops executing.

~Arrays

❖What is an array in JavaScript? How do you declare and initialize an array?

ANSWER: An array in JavaScript is a special variable used to store multiple values in a single variable. Each value in an array is called an element, and each element has an index (starting from 0).

Declaring and Initializing an Array:

1. Using square brackets [ ]

2. Using the Array constructor

❖ Explain the methods push(), pop(), shift(), and unshift() used in arrays.

ANSWER: In JavaScript, push() adds one or more elements to the end of an array, while pop() removes the last element. Similarly, shift() removes the first element from an array, and unshift() adds one or more elements to the beginning of the array.

For example, push() and unshift() are used to insert new items, while pop() and shift() are used to remove existing ones. These methods help easily manage and update array elements.

~Objects

❖What is an object in JavaScript? How are objects different from arrays?

ANSWER: In JavaScript, an object is a collection of key–value pairs used to store related data and functions. Each key is a property name (usually a string), and each value can be any data type. Objects are best for representing structured data like a person or a car. In contrast, an array is an ordered list of values accessed by numeric indexes, used to store sequences of items. Simply put, objects organize data by *name*, while arrays organize data by *position*.

❖Explain how to access and update object properties using dot notation and bracket notation.

ANSWER: In JavaScript, you can access and update object properties using dot notation or bracket notation.

Dot notation uses a period (.) followed by the property name.

Bracket notation uses square brackets ([]) and a string for the property name.

Use dot notation when the property name is a valid identifier (no spaces or special characters). Use bracket notation when the property name is stored in a variable or contains special characters or spaces.

~JavaScript Events

❖What are JavaScript events? Explain the role of event listeners.

ANSWER: JavaScript events are actions or occurrences that happen in the browser — such as a user clicking a button, typing in a text box, loading a page, or moving the mouse. JavaScript can detect these events and respond to them, allowing web pages to become interactive and dynamic.

An event listener is a function that waits for a specific event to occur on a particular element and then runs code in response. You attach an event listener using the addEventListener() method.

❖ How does the addEventListener() method work in JavaScript? Provide an example.

ANSWER: The addEventListener() method in JavaScript is used to attach an event handler to an element — it listens for a specific event (like a click or keypress) and runs a function when that event occurs.

Example:

```
const button = document.querySelector("button");

button.addEventListener("click", function() {
  console.log("Button was clicked!");
});
```

~DOM Manipulation

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

ANSWER: The DOM (Document Object Model) is a programming interface that represents the structure of a web page as a tree

of objects. Each element in an HTML document (like <p>, <div>, or <button>) becomes a node in this tree.

JavaScript interacts with the DOM to access, modify, and control the content, structure, and style of a webpage dynamically — without reloading the page.

❖ Explain the methods getElementById(), getElementsByClassName(), and querySelector() used to select elements from the DOM.

ANSWER: In JavaScript, these DOM selection methods are used to find and access HTML elements so you can read or modify them. Here's what each one does:

1. getElementById(id)

- Selects one element by its unique ID.
- Returns a single element object (or null if not found).

2. getElementsByClassName(className)

- Selects all elements with a given class name.
- Returns an HTMLCollection (a live list, not an array).

3. querySelector(selector)

- Selects the first element that matches a CSS selector (ID, class, tag, etc.).
- Very flexible — can use complex CSS selectors.

~JavaScript Timing Events (setTimeout, setInterval)

❖ Explain the setTimeout() and setInterval() functions in JavaScript. How are they used for timing events?

ANSWER: In JavaScript, setTimeout() and setInterval() are built-in functions used to execute code after a delay or repeatedly at specific time intervals — useful for creating animations, timers, or delayed actions.

◈ setTimeout()

Runs a function once after a specified number of milliseconds.

Syntax:

setTimeout(function, delay);

Example:

```
setTimeout(() => {
  console.log("Hello after 2 seconds!");
}, 2000); // 2000 ms = 2 seconds
```

🔷 setInterval()

Runs a function repeatedly at specified time intervals (in milliseconds).

Syntax:

```
setInterval(function, interval);
```

Example:

```
setInterval(() => {
  console.log("This runs every 3 seconds");
}, 3000);
```

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

ANSWER: console.log("Action will start soon...");

```
setTimeout(() => {
  console.log("This message appears after 2 seconds!");
}, 2000);
```

Output:

```
Action will start soon...
(This message appears after 2 seconds!)
```

~JavaScript Error Handling

❖What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.

ANSWER: Error handling in JavaScript is the process of detecting and managing errors that occur while your program is running — preventing the entire script from crashing. It allows you to handle unexpected situations gracefully.

JavaScript provides try, catch, and finally blocks for structured error handling.

🔹 try block

Contains code that might cause an error.

🔹 catch block

Executes if an error occurs in the try block. It receives an error object with details about the error.

🔹 finally block

(Optionally) runs after try and catch, no matter what — often used for cleanup tasks.

Example:

```
try {
  // Code that might throw an error
  let result = 10 / 0;
  console.log("Result:", result);

  // Intentional error
  console.log(unknownVariable);
}
```

```
catch (error) {
  // Handle the error
  console.log("An error occurred:", error.message);
}
finally {
  // Always runs
  console.log("Execution completed.");
}
```

Output:

```
Result: Infinity
An error occurred: unknownVariable is not defined
Execution completed.
```

❖ Why is error handling important in JavaScript applications?

ANSWER: Error handling is important in JavaScript applications because it ensures that your program can gracefully deal with unexpected problems without crashing or behaving unpredictably.

Key reasons:

1. Improves user experience – Instead of showing a broken page or stopping execution, the application can display friendly messages or fallback behavior.
2. Prevents crashes – Errors in one part of the code won't halt the entire application.
3. Helps debugging – Catching errors provides useful information (error.message or error.stack) for developers to identify and fix issues.
4. Maintains data integrity – Ensures operations like form submissions, database updates, or calculations are safely handled even if something goes wrong.
5. Supports robust applications – Especially important for complex web apps where many asynchronous actions (like network requests) can fail.
