# Module: JavaScript

### JavaScript Introduction

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

**Answer:**

* **JavaScript** is a high-level, interpreted, and lightweight programming language mainly used to make web pages **interactive and dynamic**.
* It is also known as the **scripting language of the web**, supported by all modern browsers.

**Role in Web Development:**

1. **Structure + Style + Behaviour**
   * **HTML** → provides the structure of a webpage.
   * **CSS** → provides styling and layout.
   * **JavaScript** → adds interactivity and functionality.
2. **Key Roles of JavaScript:**
   * Validating user input in forms (e.g., checking email format).
   * Creating dynamic content updates without reloading (e.g., live chat, search suggestions).
   * Manipulating HTML & CSS (e.g., showing/hiding elements, animations).
   * Working with APIs for data fetching (AJAX, Fetch API).
   * Building full-fledged web applications with frameworks like React, Angular, Vue.

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

**Answer:**

| **Feature** | **JavaScript** | **Python** | **Java** |
| --- | --- | --- | --- |
| **Type** | Scripting language, mainly for web | General-purpose, data science, AI, scripting | Object-oriented, used for enterprise apps |
| **Execution** | Runs in browsers (client-side) & Node.js (server-side) | Runs on Python interpreter | Runs on JVM (Java Virtual Machine) |
| **Syntax** | C-like, lightweight | Very simple, readable (English-like) | Verbose, strict |
| **Use Case** | Web interactivity, front-end, full-stack | Data analysis, AI, automation | Enterprise apps, Android development |
| **Typing** | Dynamically typed | Dynamically typed | Statically typed (need to declare data types) |
| **Speed** | Faster in browsers (JIT compilation) | Slower compared to JS | Faster than JS & Python (compiled bytecode) |

* **JavaScript** = best for **web development**.
* **Python** = best for **AI, ML, data science**.
* **Java** = best for **enterprise applications & Android apps**.

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

Use of <script> tag in HTML:

* The <script> tag is used to **add JavaScript** to an HTML page.
* It can contain JavaScript code directly inside the HTML, or it can link to an external JavaScript file.
* Common uses: form validation, animations, DOM manipulation, and adding interactivity to webpages.

**Linking an External JavaScript File:**

* Save your JavaScript code in a separate file (e.g., script.js).
* Use the <script> tag with the src attribute to link it:

### Variables and Data Types

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

**Answer:**

* A **variable** in JavaScript is a container used to store data values.
* Variables allow us to reuse and manipulate data in a program.

**Ways to declare variables:**

1. **var** → Old way, function-scoped, allows redeclaration & re-assignment.

var name = "abc";

1. **let** → Modern way, block-scoped, allows re-assignment but **not redeclaration**.

let age = 20;

age = 21; // ✅ allowed

1. **const** → Block-scoped, value cannot be changed (constant).

const pi = 3.14;

pi = 3.1416; ❌ Error

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

**Answer:**  
JavaScript has **two categories of data types**:

**1. Primitive Data Types (single value, immutable)**

* **String** → "Hello"
* let str = "Hello World";
* **Number** → 10, 3.14
* let Num = 25;
* **Boolean** → true / false
* let is\_active = true;
* **Undefined** → declared but not assigned a value
* let x;
* console.log(x); // undefined
* **Null** → intentional absence of value
* let y = null;

**2. Non-Primitive (Reference) Data Types**

* **Object** → collection of key-value pairs
* let person = { name: "Ayan", age: 20 };
* **Array** → ordered collection
* let fruits = ["apple", "banana", "mango"];
* **Function** → block of reusable code

function greet() { return "Hello"; }

**Question 3: Difference between Undefined and Null**

* **Undefined**: Variable declared but no value assigned (default).
* **Null**: Intentionally assigned empty value by the programmer

### JavaScript Operators

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

In JavaScript, operators are symbols used to perform operations on values and variables.

**1. Arithmetic Operators → used for mathematical calculations.**

let a = 10, b = 5;

console.log (a + b); // 15 (Addition)

console.log (a - b); // 5 (Subtraction)

console.log (a \* b); // 50 (Multiplication)

console.log (a / b); // 2 (Division)

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

console.log (a \*\* 2); // 100 (Exponentiation)

**2. Assignment Operators → used to assign values.**

let x = 10;

x += 5; // x = x + 5 → 15

x -= 3; // x = x - 3 → 12

x \*= 2; // x = x \* 2 → 24

x /= 4; // x = x / 4 → 6

**3. Comparison Operators → used to compare values (returns true/false).**

let a = 10, b = 20;

console.log(a == b); // false (equal to, checks value only)

console.log(a === b); // false (strict equal, checks value + type)

console.log(a != b); // true (not equal)

console.log(a > b); // false

console.log(a < b); // true

console.log(a >= 10); // true

console.log(b <= 20); // true

**4. Logical Operators → used with Boolean values.**

let p = true, q = false;

console.log(p && q); // false (AND → both true required)

console.log(p || q); // true (OR → at least one true)

console.log(!p); // false (NOT → reverses the value)

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

* == → checks only value (loose equality).
* === → checks value **and** type (strict equality, preferred in modern JS).

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

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

**Answer:**

* **Control flow** is the order in which statements in a program are executed.
* By default, JavaScript executes code from **top to bottom**.
* Using conditional statements like **if-else**, we can control which block of code runs based on conditions.

**How if-else works:**

* if → runs code if condition is true.
* else if → checks another condition if the first is false.
* else → runs if all conditions are false.

**Example:**

let age = 18;

if (age < 13) {

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

} else if (age >= 13 && age < 18) {

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

} else {

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

}

// Output: You are an adult

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

**Answer:**

* A **switch statement** is used to perform different actions based on multiple possible values of a variable.
* It’s often used as a cleaner alternative to writing many if-else if statements.

**Syntax:**

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");

}

// Output: Wednesday

**When to use switch over if-else?**

* Use **if-else** → when checking **ranges or complex conditions**.
* Use **switch** → when checking **one variable against many fixed values** (e.g., days, menu options, status codes).

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

**Question 1: Explain the different types of loops in JavaScript (for, while, do-while). Provide a basic example of each.**

In JavaScript, loops are used to **repeat a block of code** until a certain condition is met.

**1. for loop**

* Used when you **know in advance** how many times you want to run the loop.
* Syntax: initialization → condition → increment/decrement.

**2. while loop**

* Used when the **number of iterations is not known in advance**, and it should continue until a condition becomes false.
* Condition is **checked before** running the loop body.

**3. do-while loop**

* Similar to while, but the loop body is **executed at least once** before checking the

condition.

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

| **Feature** | **while loop** | **do-while loop** |
| --- | --- | --- |
| Condition Check | Checked **before** executing loop body | Checked **after** executing loop body |
| Minimum Execution | May run **0 times** if condition is false at the beginning | Runs **at least once**, even if condition is false |
| Use Case | When you want to execute only if condition is true | When you want to execute at least once no matter what |

## Functions

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

* A **function** is a reusable block of code that performs a specific task.
* Functions help in **code reusability, readability, and modularity**.

**Syntax (Function Declaration + Calling):**

Declaring a function

function greet(name) {

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

}

// Calling a function

greet("Ayan");

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

| **Feature** | **Function Declaration** | **Function Expression** |
| --- | --- | --- |
| Syntax | function myFunc() { } | const myFunc = function() { }; |
| Hoisting | Hoisted (can be called before it is defined) | Not hoisted (must be defined before use) |

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

**Parameters in Functions**

* **Parameters** are variables listed inside the function definition.
* They act as **placeholders** for values (called *arguments*) that are passed when calling the function.
* You can pass one, many, or no parameters.

function greet(name) { // "name" is a parameter

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

}

greet("Ayan");

**Return Values in Functions**

* A function can **return a value** using the return keyword.
* This makes the function reusable because the returned value can be stored in a variable or used in further calculations.
* If a function has no return, it returns **undefined** by default.

✅ Example:

function add(a, b) {

return a + b; // returning the result

}

let sum = add(5, 7);

console.log(sum);

## Arrays

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

* An **array** in JavaScript is a special variable that can store **multiple values in a single variable**.
* Arrays are **indexed**, meaning each element has a numeric position starting from **0**.

1. **Using square brackets (common way):**

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

1. **Using new Array():**

let numbers = new Array(10, 20, 30, 40);

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

`push()` → add at end

- `pop()` → remove from end

- `shift()` → remove from start

- `unshift()` → add at start

# Objects

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

* An **object** in JavaScript is a collection of **key–value pairs**.
* Keys are called **properties**, and values can be strings, numbers, arrays, functions, or even other objects.

✅ Example of an object:

let person = {

name: "Ayan",

age: 21,

city: "Mumbai"

};

**Difference between Objects and Arrays:**

| **Feature** | **Objects** | **Arrays** |
| --- | --- | --- |
| Data storage | Store data as **key–value pairs** | Store data as **ordered list (index-based)** |
| Access | Access with property name (e.g., person.name) | Access with index number (e.g., fruits[0]) |
| Order | Not guaranteed | Ordered (0, 1, 2, …) |
| Use case | When you need to represent real-world entities with properties | When you need a list/sequence of items |

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

**1. Dot Notation (.)**

* Used when property name is simple (no spaces, no special characters).

**2. Bracket Notation ( [ ] )**

* Used when property name has spaces, special characters, or is stored in a variable.

**Quick Recap:**

* **Object** → key-value pairs.
* **Array** → ordered list (index-based).
* **Dot notation** → obj.key.
* **Bracket notation** → obj["key"] or obj[variable].

# JavaScript Events

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

* A **JavaScript event** is an action or occurrence that happens in the browser, which JavaScript can respond to.
* Examples of events:
  + Clicking a button (click)
  + Pressing a key (keydown)
  + Loading a page (load)
  + Submitting a form (submit)

👉 **Event Listener Role:**

* An **event listener** waits for a specific event to happen and then runs a function (called an **event handler**) when the event occurs.
* It helps make web pages **interactive** and **dynamic**.

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

* The addEventListener() method is used to attach an event handler to an element **without overwriting existing events**.
* Syntax:

element. AddEventListener(event, function, use Capture);

* **Parameters:**
  + event → type of event (e.g., "click", "mouseover")
  + function → the callback function to run when the event occurs
  + useCapture (optional) → true/false for event bubbling or capturing

# DOM Manipulation

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

* The **DOM (Document Object Model)** is a programming interface that represents an **HTML or XML document** as a **tree structure**.
* Each element (like <p>, <div>, <button>) is a **node** in this tree.

👉 **How JavaScript interacts with the DOM:**

* JavaScript can **access, modify, add, or delete** elements in the DOM.
* Examples of interactions:
  + Change text or HTML content
  + Update styles (CSS)
  + Add or remove elements dynamically
  + Handle events like clicks and keypresses

Question 2: Explain the methods getElementById (), getElementsByClassName (), and query Selector() used to select elements from the DOM.

**1. getElementById ()**

* Selects an element by its **unique id**.
* Returns **a single element**.

**2. getElementsByClassName ()**

* Selects all elements with a given **class name**.
* Returns an **HTML Collection** (like an array, but not exactly).

**3. query Selector()**

* Selects the **first element** that matches a **CSS selector** (id, class, or tag).
* More flexible and powerful.

 **DOM** = tree representation of HTML.

 JavaScript can **manipulate DOM** (change text, styles, elements).

 **Selectors:**

* getElementById("id") → single element by id.
* getElementsByClassName("class") → collection of elements.
* query Selector("CSS") → first matching element using CSS selectors.

## JavaScript Timing Events (set Timeout, set Interval)

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

1. **setTimeout()**

* Executes a function **once after a specified delay** (in milliseconds).
* Syntax:

setTimeout(function, delayInMilliseconds);

* **Use case:** Delay an action, like showing a message after 3 seconds.

1. **setInterval()**

* Executes a function **repeatedly at specified intervals** (in milliseconds).
* Syntax:

setInterval(function, intervalInMilliseconds);

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

console.log("Wait for 2 seconds...");

setTimeout(function() {

console.log("2 seconds have passed!");

}, 2000); // 2000 milliseconds = 2 seconds

**Output:**

Wait for 2 seconds...

(After 2 seconds) 2 seconds have passed!

* setTimeout() → executes **once** after a delay.
* setInterval() → executes **repeatedly** at intervals.

## JavaScript Error Handling

**Question 1: What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.**

* **Error handling** in JavaScript is the process of **catching and managing runtime errors** so that the program can continue running without crashing.
* JavaScript provides **try, catch, and finally blocks** for handling errors.

**Blocks Explanation:**

1. **try**
   * Contains the code that may **throw an error**.
2. **catch**
   * Executes if an error occurs in the try block.
   * Receives the **error object** to get details about the error.
3. **finally**
   * Executes **always**, whether an error occurs or not.
   * Usually used for cleanup operations.

try {

let result = 10 / 0; // no real error here, just example

console.log("Result: " + result);

// Uncomment to simulate an error

// nonExistentFunction();

}

catch (error) {

console.log("An error occurred: " + error);

}

finally {

console.log("This block always runs.");

}

**Output:**

Result: Infinity

This block always runs.

If an actual error occurs, the catch block executes before finally.

**Question 2: Why is error handling important in JavaScript applications?**

1. Prevents the **entire program from crashing** when an error occurs.
2. Provides **meaningful feedback** to the user or developer.
3. Helps **debug and maintain** the code efficiently.
4. Ensures **critical cleanup** operations (like closing connections) are executed.

* **try** → risky code
* **catch** → handles errors
* **finally** → always executes
* **Importance** → program stability, user experience, debugging.