**JavaScript Introduction**

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

**ANS:** JavaScript is a high-level, interpreted programming language that is primarily used to create interactive and dynamic content on websites. It is one of the core technologies of the World Wide Web, alongside HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets).

### **Role of JavaScript in Web Development:**

1. **Interactivity**:

* JavaScript allows developers to create interactive elements such as dropdowns, sliders, pop-ups, form validations, and games.
* Example: When you click a button and a modal window opens, JavaScript handles that behaviour.

1. **Dynamic Content Updates**:

* JavaScript can change content on a web page without reloading it using techniques like DOM manipulation and AJAX.
* Example: Loading new comments on a blog post without refreshing the entire page.

1. **Form Validation**:

* JavaScript checks if the user's input is valid before submitting data to the server, improving user experience and reducing server load.

1. **Communication with Servers**:

* Through APIs and tools like fetch and XML Http Request, JavaScript can send/receive data from web servers in the background.

1. **Rich User Interfaces**:

* With libraries and frameworks like React, Vue.js, and Angular, JavaScript supports the creation of highly responsive and modern web applications.

**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 differ in **syntax, use cases, execution environment, and design philosophy**. Here's a comparison to highlight the key differences:

**1. Execution Environment**

* **JavaScript**:
  + Originally designed to run in **web browsers** (client-side).
  + Can now also run on **servers** using Node.js.
  + Primary use: **Web development**.
* **Python**:
  + Runs on the **server-side** or local machine.
  + Used in **web development, data science, automation, AI**, etc.
* **Java**:
  + Compiled and run on the **Java Virtual Machine (JVM)**.
  + Used for **enterprise applications, Android development, desktop apps**, etc.

**2. Syntax and Complexity**

* **JavaScript**:
  + Lightweight, flexible syntax.
  + Loosely typed (dynamic typing).
  + Easier to write quick scripts and add interactivity to web pages.
* **Python**:
  + Simple, clean, and highly readable syntax.
  + Focuses on **code readability and simplicity**.
  + Also dynamically typed.
* **Java**:
  + Verbose and strictly structured syntax.
  + Statically typed (variables must be declared with types).
  + Requires more code to accomplish the same tasks.

**3. Typing System**

* **JavaScript**: Dynamically typed
* **Python**: Dynamically typed
* **Java**: Statically typed

### **4. Use Cases**

| **Language** | **Common Use Cases** |
| --- | --- |
| **JavaScript** | Web development (frontend & backend), mobile/web apps |
| **Python** | AI, machine learning, scripting, automation, web development |
| **Java** | Android apps, large-scale enterprise systems, desktop apps |

### **5. Concurrency Model**

* **JavaScript**:
  + Uses an **event-driven, non-blocking I/O** model with an **event loop**.
  + Supports **asynchronous programming** via callbacks, promises, and async/await.
* **Python**:
  + Traditionally synchronous, though it has support for async (asyncio).
  + Thread-based concurrency is more common.
* **Java**:
  + Uses **multithreading** with built-in support for concurrency.

**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 the <script> Tag in HTML**

The <script> tag in HTML is used to **embed or reference JavaScript code** in a web page. JavaScript is used to make web pages interactive and dynamic.

##### **Key Purposes:**

* Add interactivity (e.g., forms, buttons, animations)
* Control web page behaviour (e.g., validate input)
* Load and manipulate content dynamically (e.g., fetch data from APIs)

#### **Linking an External JavaScript File**

To keep HTML clean and JavaScript reusable, it’s common to place JavaScript code in a separate .js file and link it using the src attribute of the <script> tag.

##### **Syntax:**

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

##### **Example:**

<!DOCTYPE html>

<html>

<head>

<title>External Script Example</title>

</head>

<body>

<h1>Welcome</h1>

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

</body>

</html>

**Variables and Data Types**

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

#### **ANS:**

#### **What are Variables?**

Variables in JavaScript are containers used to **store data values** (like numbers, text, or objects) that can be used and manipulated in a program.

#### **Declaring Variables:**

1. **var**
   * Function-scoped
   * Can be redeclared and updated
   * Older way of declaring variables

var name = "Alice";

1. **let**
   * Block-scoped
   * Can be updated but **not redeclared** in the same scope
   * Preferred over var for most use cases

let age = 25;

1. **const**
   * Block-scoped
   * **Cannot be updated or redeclared**
   * Used for values that should not change

const country = "India";

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

**ANS:** JavaScript has **two types of data types: primitive** and **non-primitive (object).**

#### **1. Primitive Data Types:**

* **String** – Represents text

let name = "John";

* **Number** – Represents both integers and floats

let age = 30;

* **Boolean** – Represents true or false

let isOnline = true;

* **Undefined** – A variable that has been declared but not assigned a value

let city;

* **Null** – Represents an intentionally empty value

let score = null;

* **Symbol** – Unique and immutable value, used mostly for object property keys

let sym = Symbol("id");

* **BigInt** – For very large integers

let bigNumber = 123456789012345678901234567890n;

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

* **Object** – Collection of key-value pairs

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

* **Array** – A list of values

let colors = ["red", "blue", "green"];

* **Function** – A reusable block of code

function greet() {

console.log("Hello");

}

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

**ANS:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **undefined** | **null** |
| **Meaning** | A variable declared but not assigned a value | Represents an intentional absence of value |
| **Type** | undefined (primitive) | object (primitive) |
| **Usage** | Automatically assigned by JavaScript | Manually assigned by the programmer |
| **Example** | let x; // x is undefined | let y = null; // y is null |

#### **Summary:**

* undefined means a variable **exists but has no value yet**.
* null is used to **intentionally indicate** "no value".

**JavaScript Operators**

**Question 1: What are the different types of operators in JavaScript? Explain with examples. • Arithmetic operators • Assignment operators • Comparison operators • Logical operators**

**ANS:** In JavaScript, **operators** are used to perform operations on variables and values. Here’s a breakdown of the four main types you mentioned, with clear examples:

## 1. **Arithmetic Operators**

These are used to perform basic mathematical operations.

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| + | Addition | 5 + 3 | 8 |
| - | Subtraction | 5 - 3 | 2 |
| \* | Multiplication | 5 \* 3 | 15 |
| / | Division | 6 / 3 | 2 |
| % | Modulus (Remainder) | 5 % 2 | 1 |
| \*\* | Exponentiation | 2 \*\* 3 | 8 |
| ++ | Increment | let a = 1; a++ | 2 |
| -- | Decrement | let a = 1; a-- | 0 |

## 2. **Assignment Operators**

Used to assign values to variables.

| **Operator** | **Example** | **Same As** |
| --- | --- | --- |
| = | x = 10 | — |
| += | x += 5 | x = x + 5 |
| -= | x -= 3 | x = x - 3 |
| \*= | x \*= 2 | x = x \* 2 |
| /= | x /= 2 | x = x / 2 |
| %= | x %= 3 | x = x % 3 |

## 3. **Comparison Operators**

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

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| == | Equal to (loose) | 5 == '5' | true |
| === | Equal to (strict) | 5 === '5' | false |
| != | Not equal (loose) | 5 != '5' | false |
| !== | Not equal (strict) | 5 !== '5' | true |
| > | Greater than | 6 > 3 | true |
| < | Less than | 2 < 5 | true |
| >= | Greater than or equal to | 5 >= 5 | true |
| <= | Less than or equal to | 3 <= 2 | false |

## 4. **Logical Operators**

Used to combine multiple conditions (boolean expressions).

| **Operator** | **Name** | **Example** | **Result** |
| --- | --- | --- | --- |
| && | AND | true && false | false |
| ` |  | ` | OR |
| ! | NOT | !true | false |

**Example:**

**Declaring variables**

let x = 10;

let y = 5;

**Arithmetic Operators**

console.log("Arithmetic Operators:");

console.log("x + y =", x + y); // 15

console.log("x - y =", x - y); // 5

console.log("x \* y =", x \* y); // 50

console.log("x / y =", x / y); // 2

console.log("x % y =", x % y); // 0

**Assignment Operators**

console.log("\nAssignment Operators:");

x += 5; // same as x = x + 5 → x becomes 15

console.log("x += 5 =", x);

y \*= 2; // same as y = y \* 2 → y becomes 10

console.log("y \*= 2 =", y);

**Comparison Operators**

console.log("\nComparison Operators:");

console.log("x == y:", x == y); // true (15 == 10 is false)

console.log("x != y:", x != y); // true

console.log("x > y:", x > y); // true

console.log("x < y:", x < y); // false

console.log("x === 15:", x === 15); // true

console.log("y !== 10:", y !== 10); // false

**Logical Operators**

console.log("\nLogical Operators:");

let isXGreater = x > y; // true

let isYLess = y < 20; // true

console.log("x > y && y < 20:", isXGreater && isYLess); // true

console.log("x > y || y > 20:", isXGreater || y > 20); // true

console.log("!(x < y):", !(x < y)); // true

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

### **ANS: == (Double Equals) — Loose Equality**

* Compares **values**.

### ***=== (Triple Equals) —* Strict Equality**

* Compares **both value and type, no type conversion.**

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

**ANS:** In JavaScript, undefined and null both represent the absence of a value, but they are used differently and mean different things.

### undefined:

* This means a variable has been **declared but not given a value** yet.
* It is **automatically set** by JavaScript.

**Example:**

let a;

console.log(a); // Output: undefined

### null:

* This means a variable has been **explicitly assigned "no value"**.
* It is **manually set** by the developer when you want to say “this should be empty.”

**Example:**

let b = null;

console.log(b); // Output: null

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

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

### **ANS:**

**Control flow** is the order in which code is executed in a program.

By default, JavaScript runs code **from top to bottom, line by line**. But using **control flow statements** like if-else, switch, for, and while, we can **change the flow** based on conditions.

**How if-else Statements Work:**

An if-else statement allows your code to **make decisions**.

* If a condition is **true**, it runs a block of code.
* If it’s **false**, it can run another block (with else) or skip (if only if is used).

### **Example:**

let marks = 70;

if (marks >= 90) {

console.log("Grade: A");

} else if (marks >= 75) {

console.log("Grade: B");

} else if (marks >= 50) {

console.log("Grade: C");

} else {

console.log("Fail");

}

### **Explanation:**

* The program checks if marks is 90 or more → No.
* Then it checks if marks are 75 or more → Yes → It prints **"Grade: B"**.
* Remaining conditions are skipped.

This is **control flow** in action — JavaScript decides **what to run based on conditions.**

**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 used to **test one variable or expression** against **many possible values.** It’s a cleaner alternative to writing **multiple if-else if statements** when checking the same variable.

### **How it works:**

* The switch compares the value with each case.
* If it matches, the code in that case runs.
* Use break to stop after a match. Otherwise, it will continue to the next case (called **fall-through**).
* default runs if no case matches.

### **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("Not a valid day");

}

**Output:** Wednesday

### **When to use switch instead of if-else:**

* When you are **comparing the same variable** to multiple exact values.
* It makes your code **cleaner and easier to read** than many else if blocks.

### **Use if-else when:**

* You're checking **ranges** (x > 10, x < 5).
* You're comparing **different variables or conditions**.

But for multiple exact matches like:

if (color === "red") {...}

else if (color === "blue") {...}

→ switch(color) is a better choice.

**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.**

**ANS:**

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

1️. **for loop**

* Best when you know **how many times** to loop.
* Has three parts: initialization, condition, and update.

#### **Example:**

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

console.log("Number:", i);

}

**Output:**  
Number: 1  
Number: 2  
...  
Number: 5

### **2️. while loop**

* Runs **as long as the condition is true.**
* You must manually update the variable inside the loop.

#### **Example:**

let i = 1;

while (i <= 3) {

console.log("Hello");

i++;

}

**Output:**  
Hello  
Hello  
Hello

### **3️. do-while loop**

* Runs the code **at least once**, even if the condition is false.
* The condition is checked **after** the first run.

#### **Example:**

let i = 1;

do {

console.log("Run at least once");

i++;

} while (i <= 1);

**Output:**  
Run at least once

Use for when looping a fixed number of times.

Use while when looping based on a condition.

Use do-while when you need the loop to **run at least once**, no matter what.

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

**ANS:**

The main difference is **when the condition is checked**.

**while loop:**

* **Checks the condition first**, then runs the loop body.
* If the condition is **false at the start**, the loop body **doesn’t run** at all.

let i = 5;

while (i < 3) {

console.log("Inside while loop");

}

**Output:** Nothing (condition is false from the beginning)

**do-while loop:**

* **Runs the loop body first**, then checks the condition.
* It **always runs at least once**, even if the condition is false.

let i = 5;

do {

console.log("Inside do-while loop");

} while (i < 3);

**Output:**  
Inside do-while loop

**Functions**

**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 that performs a specific task. You define a function once and can **call it anywhere, anytime** in your code.

Functions help in:

* **Organizing code**
* **Avoiding repetition**
* **Making code easier to read and maintain**

**Syntax for Declaring a Function:**

function functionName(parameters) {

// code to execute

}

* function → keyword to declare a function
* functionName → any name you choose
* parameters → optional values the function takes (inside parentheses)
* { ... } → block of code that runs when the function is called

### **Syntax for Calling a Function:**

functionName(arguments);

* You call the function by using its name followed by ()
* arguments are the actual values you pass to the function.

**Example:**

function greet(name) {

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

}

greet("Divya"); // Calling the function

**Output:**  
Hello, Divya!

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

**ANS:**

### **1)Function Declaration**

### This is the traditional way to define a function using the function keyword.

#### Syntax:

function greet() {

console.log("Hello!");

}

#### Features:

#### **Hoisted**: You can call the function **before it is defined** in the code.

#### Easy to use and understand.

#### **Example:**

greet(); // ✅ Works even before the definition

function greet() {

console.log("Hello!");}

**2️) Function Expression**

Here, the function is **assigned to a variable** (can be anonymous or named).

#### **Syntax:**

const greet = function() {

console.log("Hello!");

};

#### **Features:**

* **Not hoisted**: You **cannot** call the function before it’s defined.
* More flexible (can be used with arrow functions or passed as arguments).

#### **Example:**

greet(); // ❌ Error: Cannot access 'greet' before initialization

const greet = function() {

console.log("Hello!");

};

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

**ANS:**

Functions often need **inputs** to work with and can give back **outputs**. This is where **parameters** and **return values** come in.

**What are Parameters?**

**Parameters** are variables listed inside the parentheses **when you define** a function. They act as **placeholders for input values.**

function greet(name) {

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

}

Here, name is a **parameter.**

When you **call** the function, you pass an **argument** (real value):

greet("Divya"); // Output: Hello, Divya

**What is a Return Value?**

A function can **return** a value using the return keyword. This sends the result back to the place where the function was called.

function add(a, b) {

return a + b;

}

let result = add(5, 3);

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

* a and b are **parameters**
* 5 and 3 are **arguments**
* return a + b; sends back the result
* result stores the returned value

**Arrays**

**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** in a single place.

For example, instead of storing 5 names in 5 separate variables, you can store them all in one array.

**Why use arrays?**

* To **group related data**
* To **loop through multiple values**
* To keep code clean and organized

**How to Declare and Initialize an Array:**

#### Using square brackets [] (most common way):

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

#### **Using the Array constructor:**

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

### **Example:**

let colors = ["red", "green", "blue"];

console.log(colors[0]); // Output: red

console.log(colors.length); // Output: 3

* Arrays use **indexing** (starts from 0)
* colors[0] gives the **first element**
* length gives total number of items

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

**ANS:**

These methods help you **add or remove elements** from an array easily.

**1️) push ()**

* 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("mango");

console.log(fruits); // ["apple", "banana", "mango"]

### **2️) pop ()**

* Removes the **last element** from the array.
* Returns the removed element.

#### **Example:**

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

let lastFruit = fruits.pop();

console.log(lastFruit); // "mango"

console.log(fruits); // ["apple", "banana"]

### **3️) shift ()**

* Removes the **first element** from the array.
* Shifts remaining elements to the left.
* Returns the removed element.

#### **Example:**

let numbers = [10, 20, 30];

let firstNum = numbers.shift();

console.log(firstNum); // 10

console.log(numbers); // [20, 30]

### **4️) unshift()**

* Adds **one or more elements at the beginning** of the array.
* Returns the new length of the array.

#### **Example:**

let numbers = [20, 30];

numbers.unshift(10);

console.log(numbers); // [10, 20, 30]

**Objects**

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

**ANS:**

An **object** in JavaScript is a collection of **key-value pairs** used to store structured data.

Each key is called a **property**, and its value can be:

* A string, number, array, function, another object, etc.

### **Object Syntax:**

let person = {

name: "Divya",

age: 26,

isStudent: false

};

* name, age, isStudent → keys (also called **properties**)
* "Divya", 26, false → values

### **Accessing Object Properties:**

console.log(person.name); // "Divya"

console.log(person["age"]); // 26

### **How Objects Are Different from Arrays:**

| **Feature** | **Object** | **Array** |
| --- | --- | --- |
| Data Type | Key-value pairs | Ordered list of values |
| Access | By **keys** (e.g., obj.name) | By **index** (e.g., arr[0]) |
| Order | Not guaranteed (keys can be unordered) | Ordered (index starts at 0) |
| Use Case | Best for describing **entities** or **records** | Best for **lists or collections** of items |

### **Example Comparison:**

// Object

let car = {

brand: "Toyota",

model: "Camry",

year: 2023

};

// Array

let colors = ["red", "green", "blue"];

* Use **objects** when you need named properties (like a person, product, etc.)
* Use **arrays** when you need a list or sequence of items

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

**ANS:**

### 1️) **Dot Notation (object.property)**

### Simple and commonly used when the property name is a **valid identifier** (no spaces or special characters).

### **Example:**

let person = {

name: "Divya",

age: 26

};

// Access

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

// Update

person.age = 27;

console.log(person.age); // Output: 27

### **2️) Bracket Notation (object["property"])**

### Used when:

### The property name is **stored in a variable**

### The key contains **spaces or special characters**

### You want **dynamic access**

#### **Example:**

let person = {

name: "Divya",

"home town": "Rajula"

};

// Access

console.log(person["name"]); // Output: Divya

console.log(person["home town"]); // Output: Rajula

// Update

let key = "name";

person[key] = "Princy";

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

**JavaScript Events**

**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 and can be responded to by your code.

**Common examples:**

* Clicking a button (click)
* Typing in a field (keydown, input)
* Loading a page (load)
* Hovering over an element (mouseover)

JavaScript can **detect and respond** to these events to make web pages interactive.

### **What is an Event Listener?**

An **event listener** is a function that **waits for a specific event** to happen on an element, and then **executes a block of code** in response.

It is added using the addEventListener() method.

**Syntax:**

element.addEventListener("event", function);

* "event" → the name of the event (e.g., "click")
* function → what should happen when the event occurs

**Example:**

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

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

alert("Button clicked!");

});

* When the button is clicked, the alert pops up.

**Why Use Event Listeners?**

* Keeps HTML and JavaScript separate
* Allows multiple listeners on the same element
* Helps make websites **dynamic and interactive**

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

**ANS:**

The addEventListener() method allows you to **attach a function** (called a **callback**) to a specific **event** on an HTML element.  
When that event occurs, the attached function is executed.

**Syntax:**

element.addEventListener("event", function);

* element → The HTML element you're targeting (like a button or input)
* "event" → The name of the event (e.g., "click", "mouseover", "keydown")
* function → The function to run when the event happens

**Example:**

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

<script>

let btn = document.getElementById("myBtn");

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

alert("You clicked the button!");

});

</script>

**DOM Manipulation**

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

**ANS:**

The **DOM (Document Object Model)** is a **tree-like structure** created by the browser to represent your **HTML page.**

* Every **HTML element** (like <h1>, <p>, <div>) becomes a **node** in the tree.
* JavaScript uses the DOM to **read, change, add, or delete** HTML elements on a web page.

Think of the DOM as a **bridge between HTML and JavaScript.**

**How JavaScript Interacts with the DOM:**

1. **Access elements**
2. **Change content**
3. **Change styles**
4. **Add/remove elements**
5. **Respond to events**

### **1. Accessing Elements:**

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

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

### **2. Changing Content:**

heading.textContent = "New Title";

### **3. Changing Styles:**

heading.style.color = "blue";

### **4. Creating & Adding Elements:**

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

newPara.textContent = "Hello DOM!";

document.body.appendChild(newPara);

### **5. Handling Events:**

let btn = document.getElementById("myBtn");

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

alert("Button clicked!");

});

**Question 2**: **Explain the methods getElementById(), getElementsByClassName(),and**

**querySelector() used to select elements from the DOM.**

**ANS:**

### **1) getElementById()**

* Selects a **single element** by its **id**.
* Returns **one element** or null if not found.

Example:

<h1 id="title">Welcome</h1>

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

console.log(heading.textContent); // Output: Welcome

### **2️) getElementsByClassName()**

* Selects **all elements** with a given **class name.**
* Returns an **HTML Collection** (like an array but not exactly).
* You need to loop through it to access each item.

**Example:**

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

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

let paras = document.getElementsByClassName("info");

console.log(paras[0].textContent); // Output: Paragraph 1

### **3️) querySelector()**

* Returns the **first element** that matches a **CSS-style selector.**
* More flexible than getElementById or getElementsByClassName.

#### **Example:**

<div class="box" id="mainBox"></div>

let box = document.querySelector("#mainBox"); // By ID

let firstBox = document.querySelector(".box"); // By class

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

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

**ANS:**

### **setTimeout() and setInterval() in JavaScript**

Both setTimeout() and setInterval() are **timing functions** in JavaScript used to execute code **after a delay** or **repeatedly at intervals.**

**setTimeout()**

* Executes a function **once after a specified delay** (in milliseconds).
* Used for **delayed execution.**

#### **Syntax:**

setTimeout(function, delay);

#### **Example:**

setTimeout(function() {

console.log("Hello after 2 seconds");

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

Runs the message **once after 2 seconds.**

**setInterval()**

* Executes a function **repeatedly at fixed intervals**.
* Keeps running until stopped using clearInterval().

#### **Syntax:**

setInterval(function, interval);

#### **Example:**

setInterval(function () {

console.log("Repeating every 1 second");

}, 1000); // Every 1000 ms = 1 second

Runs the message **every second** continuously.

Stopping Them:

* **Stop setTimeout** (if stored in a variable):

let timer = setTimeout(...);

clearTimeout(timer);

* **Stop setInterval**:

let repeat = setInterval(...);

clearInterval(repeat);

### **Use Cases:**

* setTimeout() → show a message after delay, animations, hide alerts
* setInterval() → clocks, auto-refresh, repeated animations or updates

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

**ANS:**

Here’s a simple example that shows a message **after a 2-second delay:**

console.log("Action will happen in 2 seconds...");

setTimeout(function() {

console.log("⏰ Time's up! This message is delayed by 2 seconds.");

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

### **How it Works:**

1. The first console.log() runs immediately.
2. setTimeout() waits for **2000 milliseconds**.
3. After that, it runs the code inside the function.

You can replace the console.log() with any action (like showing an alert, hiding a message, etc.).

**JavaScript Error Handling**

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

**ANS:**

**Error handling** in JavaScript is a way to **detect and manage errors** during program execution **without crashing** the entire script.

JavaScript provides the try, catch, and finally blocks to **handle runtime errors gracefully.**

**try...catch...finally Structure:**

try {

// Code that may throw an error

} catch (error) {

// Code to handle the error

} finally {

// Code that always runs (optional)

}

### **Explanation of Each Block:**

* **try**:  
  Code that might throw an error is placed here.
* **catch**:  
  If an error occurs in try, this block runs. The error object gives details.
* **finally** (optional):  
  Always runs — whether there was an error or not. Used for cleanup or final steps.

### **Example:**

try {

let num = 10;

let result = num.toUpperCase(); // Error: toUpperCase not valid on number

console.log(result);

} catch (err) {

console.log(" An error occurred:", err.message);

} finally {

console.log(" This runs no matter what.");

}

**Output:**

An error occurred: num.toUpperCase is not a function

This runs no matter what.

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

**ANS:**

Error handling is **crucial** because it helps your application deal with problems **gracefully** instead of **crashing** or behaving unpredictably.

Here’s **why it matters:**

### 1️) **Prevents Application Crashes**

Without error handling, one small error (like dividing by zero or accessing undefined) can stop the whole script.

With proper handling: The error is caught, and the app continues running.

2️) **Improves User Experience**

Instead of showing a broken page or technical error, you can show:

* A friendly error message (e.g., "Something went wrong!")
* Retry options or fallback actions

This keeps users informed and comfortable.

### 3) **Helps Debugging**

Using catch blocks and console.log(error) gives:

* Clear information about **what went wrong**
* Which **line or part** of code caused the issue

This makes bugs **easier to find and fix.**

### 4️) **Essential for Network & API Calls**

When working with:

* Fetch requests
* Async/await
* APIs or databases

Errors like “network failure” or “server not found” must be handled to avoid blank screens or silent failures.

### 5️) **Supports Clean-Up Tasks**

With finally, you can:

* Close open files
* Reset loading indicators
* Disconnect from a database, etc.

Even if an error occurs, clean-up is guaranteed.

Top of Form

Bottom of Form