**Q-1) What is JavaScript? Explain the role of JavaScript in web development.**

* JavaScript is the language that makes a website **"do things"** — like showing messages, changing colors , playing videos, sliding images, or responding when you click a button.
* Example: When you click a **"Submit"** button on a form and the page checks if you filled it correctly **without reloading** — that’s JavaScript.
* Role of JavaScript in Web Development:

So JavaScript :

* Adds **interactivity** (e.g., buttons, menus, sliders)
* Updates content **without reloading** the page (used in social media feeds)
* Validates forms (e.g., checking if an email is valid)
* Builds complex features like chat apps, games, or interactive maps

**Q-2) How is JavaScript different from other programming languages like Python or Java?**

1. Where They Are Used :-

* JavaScript is mostly used in web development — it runs in the browser to make websites interactive.
* Python is used for data science, automation, AI, backend development, etc.
* Java is often used for mobile apps (especially Android), large systems, and enterprise software.

2. Where the Code Runs :-

* JavaScript runs in the browser (like Chrome or Firefox).
* Python and Java run on your computer/server, not in the browser.

3. Ease of Learning :-

* Python is the easiest to learn. It reads like English.
* JavaScript is also beginner-friendly but a bit more tricky due to browser behaviour.
* Java is more strict, with longer code and rules.

4. Main Strength :-

* JavaScript → Great for making webpages interactive.
* Python → Great for solving real-world problems and doing AI/data.
* Java → Great for large-scale, powerful applications.

**Q-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 in HTML to add JavaScript code to a webpage.
* You can either:
* Write the JavaScript directly inside the HTML
* Or link an external file
* Let’s say you have a JavaScript file named script.js.
* You can link it like this:

<!DOCTYPE html>

<html>

<body>

<h1>Hello</h1>

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

</body>

</html>

* In this case :
* JavaScript is written in a separate file (script.js)
* The src inside <script> tells HTML where to find the file
* This is called external JavaScript

**Q-4) What are variables in JavaScript? How do you declare a variable using var, let, and const?**

* A variable is like a box where you can storedata — such as numbers, text, or anything else — so you can use it later in your code.
* Imagine a box labeled name that stores the word "Kaushal" — that’s a variable.
* In JavaScript, you can create a variable using :
* var
* let
* const

**1. var**

* This is the **old way** of declaring variables.
* Example :-

var name = "Kaushal";

* You **can change** the value later.
* Used in older JavaScript code.
* Less strict — can lead to unexpected bugs.

**2. let**

* This is the **modern** and safer way.
* Example :-

let age = 25;

* You **can change** the value later.
* Better to use than var for most cases.
* Block-scoped (safer inside functions/loops)

**3. const**

* This is used when the value **should not change**.
* Example :-

const country = "India";

* You **cannot change** the value after it's set.
* Good for fixed values (like days of the week, pi, etc.)

**Q-5) Explain the different data types in JavaScript. Provide examples for each.**

* In JavaScript, datatypes are the different kinds of values you can use — like numbers, text, true/false, etc.
* Let’s go over them in a simpleandclear way with examples.

**1. Number**

* + Used for any kind of number — whole or decimal.
  + Example :-

let age = 25;

let price = 99.99;

**2. String**

* Text or characters inside quotes ("" or '').
* Example :-

let name = "Kaushal";

let message = 'Hello World';

**3. Boolean**

* Only two values: true or false.
* Example :-

let isStudent = true;

let has License = false;

**4. Undefined**

* Means the variable exists but no value is given yet.
* Example :-

let result;

console.log(result); // undefined

**5. Null**

* It means nothing or emptyonpurpose.
* Example :-

let data = null;

**6. Object**

* Used to store many values together in a single variable using key–valuepairs.
* Example :-

let person = {

name: "Kaushal",

age: 25,

isStudent: true

};

**7. Array**

* A type of object used to store lists of values.
* Example :-

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

**Q-6) What is the difference between undefined and null in JavaScript?**

* In simple words, undefined and null both mean “nothing”, but they are used differently in JavaScript.

1. **Undefined**

* Happens automatically when you don’t assign a value to a variable.
* JavaScript says: “This variable exists, but I don’t know what it is yet.”
* Example :-

let name;

console.log(name); // undefined

1. **null**

* You set it yourself when you want to say “this is empty on purpose.”
* JavaScript says: “I know this is nothing, and that’s OK.”
* Example :-

let age = null;

console.log(age); // null

**Q-7) What are the different types of operators in JavaScript? Explain with examples.**

**• Arithmetic operators**

**• Assignment operators**

**• Comparison operators**

**• Logical operators**

**1. Arithmetic Operators (Used for math)**

* These are used to do calculations like add, subtract, multiply, etc.

| **Operator** | **Meaning** | **Example** | **Result** |
| --- | --- | --- | --- |
| + | Add | 5 + 3 | 8 |
| - | Subtract | 10 - 4 | 6 |
| \* | Multiply | 6 \* 2 | 12 |
| / | Divide | 8 / 2 | 4 |
| % | Remainder | 10 % 3 | 1 |

**2. Assignment Operators (Used to store value in a variable)**

* These are used to assign values to variables.

| **Operator** | **Meaning** | **Example** | **Final Value of x** |
| --- | --- | --- | --- |
| = | Assign | let x = 10 | 10 |
| += | Add and assign | x += 5 | 15 (10+5) |
| -= | Subtract and assign | x -= 3 | 7 (10–3) |
| \*= | Multiply and assign | x \*= 2 | 20 (10×2) |
| /= | Divide and assign | x /= 2 | 5 (10÷2) |

**3. Comparison Operators (Used to compare values)**

* These are used to check conditions. They give results like true or false.

| **Operator** | **Meaning** | **Example** | **Result** |
| --- | --- | --- | --- |
| == | Equal (value) | 5 == "5" | true |
| === | Equal (value and type) | 5 === "5" | false |
| != | Not equal | 5 != 3 | true |
| > | Greater than | 10 > 5 | true |
| < | Less than | 3 < 2 | false |
| >= | Greater than or equal to | 6 >= 6 | true |
| <= | Less than or equal to | 7 <= 9 | true |

**4. Logical Operators (Used to combine conditions)**

* These are used when you want to check multiple things at once.

| **Operator** | **Meaning** | **Example** | **Result** |
| --- | --- | --- | --- |
| && | AND (both true) | 5 > 3 && 10 > 5 | true |
| ` |  | ` | OR (at least one true) |
| ! | NOT (opposite) | !true | false |

**Q-8) What is the difference between == and === in JavaScript?**

| **Feature** | **== (Double Equals)** | **=== (Triple Equals)** |
| --- | --- | --- |
| Name | Loose Equality | Strict Equality |
| Checks | Only values | Values + DataTypes both |
| Type Conversion | Yes (It converts types if needed) | No (No conversion, types must match) |
| Example 1 | 5 == "5" → true | 5 === "5" → false |
| Example 2 | 0 == false → true | 0 === false → false |
| When to Use | Not recommended (can cause bugs) | Better and safer to use |

**Q-9) What is control flow in JavaScript? Explain how if-else statements work with an example.**

* **Control flow** in JavaScript means the order in which the computer runs your code. JavaScript normally runs code **line by line**, from top to bottom. \
* But sometimes, we want to run some code only **if a certain condition is true** — that’s where **if-else** comes in.
* The **if-else statement** helps us **make decisions** in the code.
* It checks a condition:
* If the condition is **true**, it runs one block of code.
* If it is **false**, it can run a different block (using else).
* **Example :**

let age = 20;

if (age >= 18) {

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

} else {

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

}

* **Output :**  
  You are an adult.  
  Because age is 20 which is greater than or equal to 18.

**Q-10) Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?**

* A **switch statement** is used when you want to compare **one value with many options**.
* It is an **alternative to using many if-else statements**.
* In a switch, we:
* Match a value with different **cases**.
* Use **break** to stop the execution when the correct case is found.
* Use **default** if no case matches.
* **Example :**

let day = "Tuesday";

switch (day) {

case "Monday":

console.log("Start of the week!");

break;

case "Tuesday":

console.log("Second day of the week.");

break;

case "Friday":

console.log("Weekend is near!");

break;

default:

console.log("Just another day.");

}

* **Output :**  
  Second day of the week.

Because the value of day is "Tuesday".

* **When to use switch instead of if-else?**
* Use **if-else** when you are checking **ranges or conditions** (like age > 18).
* Use **switch** when you are comparing **one variable with many fixed values** (like days, months, colors, etc.).

**Q-11) Explain the different types of loops in JavaScript (for, while, do-while). Provide a basic example of each.**

* Loops are used to **repeat a block of code** multiple times until a condition becomes false.

**1. for loop :**

* The for loop runs a block of code a specific number of times.
* It has **3 parts**: start, condition, and update.

**Example:**

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

console.log("Number is: " + i);

}

**Output :**

Number is: 1

Number is: 2

Number is: 3

Number is: 4

Number is: 5

**2. while loop :**

* The while loop runs the code **as long as** the condition is true.
* It checks the condition **before** running the loop.

**Example :**

let i = 1;

while (i <= 3) {

console.log("Count: " + i);

i++;

}

**Output :**

Count: 1

Count: 2

Count: 3

**3. do-while loop :**

* The do-while loop **runs the code once first**, and then checks the condition.
* It always runs **at least one time**, even if the condition is false.

**Example :**

let i = 1;

do {

console.log("Hello " + i);

i++;

} while (i <= 2);

**Output :**

Hello 1

Hello 2

**Q-12) What is the difference between a while loop and a do-while loop?**

| **Feature** | **while loop** | **do-while loop** |
| --- | --- | --- |
| **Condition Check** | Checks condition **before** executing the loop | Checks condition **after** executing the loop |
| **Minimum Execution** | May **not run at all** if condition is false | **Always runs at least once** |
| **Syntax Position** | Condition comes **at the beginning** | Condition comes **at the end** |
| **Use Case** | Use when you want to run code **only if** condition is true | Use when code must run **at least once**, no matter what |

**Q-13) What are functions in JavaScript? Explain the syntax for declaring and calling a function.**

* **Functions** are reusable blocks of code in JavaScript.
* They help you write code once and use it again and again.
* A function performs a specific task when you **call (use)** it.

**🔹 Declaring (Creating) a Function:**

function greet() {

console.log("Hello!");

}

**🔹 Calling (Using) a Function:**

greet(); // Output: Hello!

* You can create functions for any task like doing calculations, showing messages, etc.

**Q-14) What is the difference between a function declaration and a function expression?**

* Both are ways to define functions, but they work slightly differently.

| **Feature** | **Function Declaration** | **Function Expression** |
| --- | --- | --- |
| **Syntax** | function greet() { ... } | const greet = function() { ... }; |
| **Hoisting** | ✅ Hoisted (You can call it before it's defined) | ❌ Not hoisted (Must be defined before use) |
| **Name** | Has a name | Can be anonymous or named |
| **Use in Variables** | Not stored in a variable | Stored in a variable |

**Q-15) Discuss the concept of parameters and return values in functions.**

* **Parameters :**
* Parameters are like **placeholders** inside a function.
* They allow you to **pass data** into the function.

Example:

function greet(name) {

console.log("Hello " + name);

}

greet("Rahul"); // Output: Hello Rahul

* Here, name is a **parameter**, and "Rahul" is the **argument** passed to it.
* **Return Values:**
* A function can **return a value** using the return keyword.
* That value can be used later in your code.

Example:

function add(a, b) {

return a + b;

}

* let result = add(5, 3); // result is 8

**Q-16) What is an array in JavaScript? How do you declare and initialize an array?**

* An **array** is a special type of variable in JavaScript that can **store multiple values** in a single place.
* For example, instead of creating 3 different variables for 3 fruits, you can store them all in one array.

**🔹 Declaring and Initializing an Array:**

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

* In the above example:
* fruits is the name of the array.
* "Apple", "Banana", and "Mango" are the **elements** of the array.
* The values are stored in **square brackets []** and separated by **commas**.
* You can also create an empty array like this:

let numbers = [];

**Q-17) Explain the methods push(), pop(), shift(), and unshift()used in arrays.**

* These are **built-in methods** used to **add or remove elements** in arrays.

| **Method** | **What It Does** | **Example** | **Resulting Array** |
| --- | --- | --- | --- |
| **push()** | Adds an element **at the end** | fruits.push("Orange") | ["Apple", "Banana", "Orange"] |
| **pop()** | Removes the **last element** | fruits.pop() | ["Apple", "Banana"] |
| **shift()** | Removes the **first element** | fruits.shift() | ["Banana", "Mango"] |
| **unshift()** | Adds an element **at the beginning** | fruits.unshift("Grapes") | ["Grapes", "Apple", "Banana"] |

**Q-18) What is an object in JavaScript? How are objects different from arrays?**

* An **object** in JavaScript is a special type of variable that stores **data in key-value pairs**.
* It is used to group related data together.

**🔹 Example of an object:**

let person = {

name: "Rahul",

age: 25,

city: "Delhi"

};

* Here,
* name, age, and city are **keys (also called properties)**
* "Rahul", 25, and "Delhi" are their **values**
* **Difference between Arrays and Objects:**

| **Feature** | **Array** | **Object** |
| --- | --- | --- |
| **Stores** | A list of values (ordered) | Key-value pairs (unordered) |
| **Indexing** | Uses **numbers** as index (0, 1, 2...) | Uses **keys** as index ("name", etc.) |
| **Use Case** | For lists like fruits, numbers, etc. | For structured data like person info |

**Q-19) Explain how to access and update object properties using dot notation and bracket notation.**

* You can get or change the value of an object property in two ways:

**1. Dot Notation (most common way):**

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

// Access

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

// Update

person.age = 26;

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

**2. Bracket Notation (used when key is in a variable or has spaces):**

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

// Access

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

// Update

person["age"] = 30;

console.log(person["age"]); // Output: 30

* You can also use bracket notation when the key is stored in a variable:

let key = "name";

console.log(person[key]); // Output: Rahul

**Q-20) What are JavaScript events? Explain the role of event listeners.**

* **JavaScript events** are actions or happenings that occur in the browser — like:
* A button being clicked
* A key being pressed
* A page being loaded
* A mouse moving over an element
* These events let JavaScript **respond to user actions** on a web page.

**🔹 What is an Event Listener?**

* An **event listener** is like a watcher.
* It **waits for an event to happen**, and when it happens, it runs a function (your code).
* It helps you add interactivity to your webpage.

**Q-21) How does the addEventListener()method work in JavaScript? Provide an example.**

* The addEventListener() method is used to **listen for a specific event** on an HTML element and then **run a function** when that event occurs.

**🔹 Syntax:**

element.addEventListener("event", function);

* element → the HTML element (like a button)
* "event" → the type of event (like "click", "mouseover", "keydown")
* function → the code that should run when the event happens

**🔸 Example:**

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

<script>

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

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

alert("Button was clicked!");

});

</script>

**What happens here:**

* The browser waits for the **click** event on the button.
* When the user clicks it, the function runs and shows an alert.

**Q-22) What is the DOM (Document Object Model) in JavaScript? How does JavaScriptinteract with the DOM?**

* The DOM (Document Object Model) is like a tree structure that represents all the elements on a web page — like paragraphs, buttons, images, etc.
* It allows JavaScript to access, change, add, or remove elements on the page.
* Think of the DOM as the bridge between JavaScript and the webpage.

**Q-23) Explain the methods getElementById(), getElementsByClassName(),and querySelector()used to select elements from the DOM.**

* These methods help you **find specific HTML elements** on the page so you can work with them in JavaScript.

| **Method** | **What it selects** | **Returns** | **Example** |
| --- | --- | --- | --- |
| **getElementById()** | Selects one element by its **unique ID** | A single element object | document.getElementById("header") |
| **getElementsByClassName()** | Selects all elements with a specific **class name** | A collection (array-like) of elements | document.getElementsByClassName("menu-item") |
| **querySelector()** | Selects the **first element** that matches a CSS selector | A single element object | document.querySelector(".button-primary") |

**Q-24) Explain the setTimeout() and setInterval() functions in JavaScript. How are they used for timing events?**

* **setTimeout()** runs a piece of code **once after a delay** (in milliseconds).
* Use it when you want to **perform an action after waiting** for some time.

**Example of setTimeout():**

setTimeout(function() {

console.log("This message shows after 3 seconds");

}, 3000); // waits 3 seconds before running the code

* **setInterval()** runs a piece of code **repeatedly at regular intervals** (in milliseconds).
* Use it when you want to **repeat an action over and over** every fixed time.

**Example of setInterval():**

setInterval(function() {

console.log("This message shows every 2 seconds");

}, 2000); // runs every 2 seconds repeatedly

**Q-25) Provide an example of how to use setTimeout()to delay an action by 2 seconds.**

* The **setTimeout()** function lets you run a block of code **after a certain amount of time has passed**.
* The time delay is given in **milliseconds** (1000 milliseconds = 1 second).
* **Example: Delay action by 2 seconds**

setTimeout(function() {

console.log("Hello! This message appears after 2 seconds");

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

**What happens here:**

* When JavaScript reaches this code, it **sets a timer** for 2 seconds.
* Meanwhile, JavaScript can keep doing other tasks.
* After 2 seconds, the message "Hello! This message appears after 2 seconds" is printed on the console.
* The function runs **only once** after the delay.

**Q-26) What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.**

* **Error handling** means writing code in such a way that if something goes wrong (an error occurs), the program doesn’t crash.
* Instead, we can **catch the error**, show a message, or handle it in a safer way.
* JavaScript provides the **try, catch, and finally** blocks to handle errors.
* **Parts of Error Handling:**

1. **try block**
   * You write the code that **might cause an error** inside try.
2. **catch block**
   * If an error happens in the try block, the catch block runs.
   * It helps you **handle the error gracefully**.
3. **finally block (optional)**
   * This block **always runs**, whether there was an error or not.
   * It is useful for cleanup work, like closing a file, hiding a loader, etc.

* **Example:**

try {

let result = 10 / 0;

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

let name = undefined;

console.log(name.length); // This will cause an error

} catch (error) {

console.log("Something went wrong: " + error.message);

} finally {

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

}

* **What happens here:**
* JavaScript tries to run all the code in the try block.
* When it reaches name.length, it throws an error because name is undefined.
* The catch block runs and shows the error message.
* The finally block runs **no matter what**.

**Q-27) Why is error handling important in JavaScript applications?**

* **Error handling is very important** because it keeps your application **safe, user-friendly, and professional.**
* **Reasons why error handling is important:**

1. **Prevents app from crashing**
   * Without error handling, even a small bug can stop the whole app from working.
2. **Gives better user experience**
   * You can show friendly error messages instead of technical crashes.
3. **Helps in debugging**
   * catch blocks can log error details to the console or send them to a server for developers.
4. **Ensures code continues smoothly**
   * Even if one part fails, the rest of the program can keep running.
5. **Works well with real-world problems**
   * For example: missing data from the server, user entering wrong input, or network failure.