**🎯 JavaScript Interview Prep Course for Freshers**

**👶 Module 1: JavaScript Fundamentals – *“The Basics They Always Ask”***

**📌 1.1 What is JavaScript?**

**💬 Interviewer might ask:**

"What is JavaScript and where is it used?"

**✅ Answer (speak naturally like this):**

JavaScript is a scripting language used to make web pages interactive. It runs in the browser, and now also on the server using Node.js. It’s mostly used for things like buttons, forms, popups, or updating content dynamically.

**📌 1.2 How is JavaScript different from Java?**

**💬 Interviewer might ask:**

"Is JavaScript related to Java?"

**✅ Answer:**

No, they’re completely different. Java is a programming language, while JavaScript is a scripting language. The names sound similar, but they serve different purposes.

**📌 1.3 Is JavaScript interpreted or compiled?**

**✅ Answer:**

JavaScript is an interpreted language. The browser reads and runs it line by line using an engine like V8 (in Chrome).

**📌 1.4 Where do you write JavaScript?**

**✅ Answer:**

JavaScript can be written inside HTML files using <script> tags, or in separate .js files and linked to HTML.

✅ **Example:**

html

CopyEdit

<!-- Inline JavaScript -->

<script>

alert("Hello from JS!");

</script>

<!-- External JS -->

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

**📌 1.5 Is JavaScript case-sensitive?**

✅ **Answer**:

Yes, myVar and MyVar are different. JavaScript is case-sensitive.

✅ **Example:**

js

CopyEdit

let name = "Aman";

let Name = "Ravi";

console.log(name); // Aman

console.log(Name); // Ravi

**📌 1.6 How do you write a comment in JavaScript?**

✅ **Example:**

js

CopyEdit

// This is a single-line comment

/\*

This is a

multi-line comment

\*/

**📌 1.7 What are variables in JavaScript?**

✅ **Answer**:

A variable is used to store data. You declare it using var, let, or const.

✅ **Example:**

js

CopyEdit

var name = "Riya";

let age = 22;

const city = "Delhi";

**📌 1.8 What is the difference between var, let, and const?**

**💬 Interviewer may ask:**

"Can you explain how var, let, and const are different?"

✅ **Answer (short and safe for freshers):**

var is function-scoped, while let and const are block-scoped.  
Also, const cannot be reassigned.

✅ **Example:**

js

CopyEdit

function test() {

if (true) {

var x = 10;

let y = 20;

const z = 30;

}

console.log(x); // ✅ Works (function scope)

// console.log(y); ❌ Error

// console.log(z); ❌ Error

}

**📌 1.9 How do you check the type of a variable?**

✅ **Answer**:

You use the typeof operator.

✅ **Example:**

js

CopyEdit

let x = 10;

console.log(typeof x); // number

**📌 1.10 What are some basic data types in JS?**

✅ **List to remember:**

* String
* Number
* Boolean
* Null
* Undefined
* BigInt
* Symbol

✅ **Example:**

js

CopyEdit

let str = "hello"; // String

let num = 42; // Number

let isTrue = true; // Boolean

let empty = null; // Null

let notSet; // Undefined

**📚 Summary for Module 1:**

| **Concept** | **Example / Note** |
| --- | --- |
| JS Use | Make pages interactive |
| Case-sensitive | Yes (a != A) |
| Comments | // or /\* \*/ |
| Declaring variables | var, let, const |
| Scoping difference | var → function; let/const → block |
| Data types | String, Number, Boolean, Null, Undefined |
| Type checking | typeof |

🔚 **End of Module 1**  
When you're ready, say: **Next module** — and we’ll go to Module 2: Operators, Expressions & Type Coercion (most commonly asked in freshers' rounds too).

**📌 2.1 What are the different types of operators in JavaScript?**

✅ **Answer (just list for interviews):**

JavaScript has:

* Arithmetic Operators (+, -, \*, /, %)
* Assignment Operators (=, +=, -=, etc.)
* Comparison Operators (==, ===, !=, !==, <, >, etc.)
* Logical Operators (&&, ||, !)
* Unary Operators (typeof, ++, --)
* Ternary Operator (? :)

**📌 2.2 Difference between == and ===**

**💬 Interviewer may ask:**

"What is the difference between == and ===?"

✅ **Answer**:

== checks only **value** (type conversion happens),  
=== checks both **value and type** (no conversion).

✅ **Example:**

js

CopyEdit

console.log(5 == "5"); // true (type coercion)

console.log(5 === "5"); // false (different types)

✅ Just remember:

👉 Use === always to avoid bugs

**📌 2.3 Output-based: null and undefined with == and ===**

✅ **Example:**

js

CopyEdit

console.log(null == undefined); // true

console.log(null === undefined); // false

✅ Why?

== converts types → treats null and undefined as equal  
=== compares types → they're different

**📌 2.4 What will this print?**

js

CopyEdit

console.log("5" - 1); // ?

console.log("5" + 1); // ?

✅ **Answer:**

js

CopyEdit

console.log("5" - 1); // 4 → because "-" converts to number

console.log("5" + 1); // "51" → "+" with string = concatenation

👉 **Common concept**:

JavaScript converts types depending on the operator (this is **type coercion**)

**📌 2.5 Type Coercion – what is it?**

✅ **Answer:**

It’s when JavaScript automatically converts one data type to another while evaluating expressions.

✅ Example:

js

CopyEdit

console.log(true + 1); // 2 (true → 1)

console.log(false + 1); // 1 (false → 0)

console.log("4" \* "2"); // 8 (both get converted to number)

**📌 2.6 Ternary Operator – very common for MCQ/interview**

✅ **Syntax:**

js

CopyEdit

condition ? valueIfTrue : valueIfFalse;

✅ Example:

js

CopyEdit

let age = 18;

let result = age >= 18 ? "Adult" : "Minor";

console.log(result); // "Adult"

**📌 2.7 Logical Operators – Interviewers LOVE && and || output Qs**

**💬 Interviewer may ask:**

"What does this return?"

js

CopyEdit

console.log(0 || "Hello"); // ?

console.log("Hi" && "Bye"); // ?

✅ **Answer:**

js

CopyEdit

0 || "Hello" → "Hello" (returns first truthy)

"Hi" && "Bye" → "Bye" (returns last truthy)

✅ Short rule:

* || → returns first **truthy**
* && → returns first **falsy** or last truthy

**📌 2.8 Pre and Post Increment (++, --)**

✅ **Example:**

js

CopyEdit

let a = 5;

console.log(a++); // 5 (then becomes 6)

console.log(++a); // 7 (already incremented)

👉 These small things are **favorite in MCQs and output rounds**.

**📌 2.9 What is typeof NaN?**

✅ **Answer:**

"number" — weird but true.

✅ Example:

js

CopyEdit

console.log(typeof NaN); // "number"

✅ Note:

NaN means “Not a Number” but still its type is "number" — **trick question**.

**📌 2.10 What is typeof null?**

✅ **Answer:**

"object" — it's a **known bug** in JavaScript.

✅ Example:

js

CopyEdit

console.log(typeof null); // "object"

**✅ Summary Table: Quick Review**

| **Concept** | **Example / Output** |
| --- | --- |
| == vs === | "5" == 5 → true, "5" === 5 → false |
| Type coercion | "5" + 1 → "51", "5" - 1 → 4 |
| null == undefined | true |
| Ternary operator | age >= 18 ? "Adult" : "Minor" |
| Logical ops | `0 |
| typeof NaN | "number" |
| typeof null | "object" |

🧩 **Mini Challenge** (Optional, for confidence): What will be the output of these?

js

CopyEdit

console.log(null + 1);

console.log(true + false);

console.log("2" \* "3");

console.log("10" - "2" + "5");

# 🚀 ****Module 3: Functions in JavaScript (Freshers Interview Prep)****

Functions are **core to JS**, and interviewers love to ask **definition + syntax + output + concept confusion** style questions. We’ll cover all that step-by-step.

## 📌 3.1 What is a function in JavaScript?

✅ **Short Interview Answer**:

A function is a block of code designed to perform a particular task. It runs when it's called.

✅ **Example**:

js

CopyEdit

function greet() {

console.log("Hello!");

}

greet(); // "Hello!"

## 📌 3.2 Function Declaration vs Function Expression

### 💬 Interviewer may ask:

"What is the difference between function declaration and function expression?"

✅ **Function Declaration** (Hoisted)

js

CopyEdit

greet(); // Works!

function greet() {

console.log("Hi!");

}

✅ **Function Expression** (Not Hoisted)

js

CopyEdit

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

const greet = function () {

console.log("Hi!");

};

🧠 **Remember**:

Declarations are hoisted, expressions are not.

## 📌 3.3 What are Parameters vs Arguments?

✅ **Short Answer**:

Parameters are variables in the function definition.  
Arguments are the actual values passed when calling the function.

✅ **Example**:

js

CopyEdit

function sum(a, b) { // a, b are parameters

return a + b;

}

sum(5, 3); // 5, 3 are arguments

## 📌 3.4 What is the return keyword?

✅ **Short Answer**:

It ends function execution and specifies the value to be returned.

✅ Example:

js

CopyEdit

function getName() {

return "ChatGPT";

}

console.log(getName()); // "ChatGPT"

## 📌 3.5 Anonymous Functions (often asked)

✅ **Short Answer**:

Functions without a name, usually used as expressions.

✅ Example:

js

CopyEdit

const greet = function() {

console.log("Hello");

};

greet();

## 📌 3.6 Arrow Functions — always asked now

✅ **Syntax**:

js

CopyEdit

const add = (a, b) => a + b;

console.log(add(2, 3)); // 5

🧠 Used more in modern JS. Shorter syntax.

## 📌 3.7 Immediately Invoked Function Expression (IIFE)

✅ **Short Answer**:

A function that runs immediately after it's defined.

✅ Example:

js

CopyEdit

(function () {

console.log("IIFE ran!");

})(); // "IIFE ran!"

🧠 Used to avoid polluting global scope.

## 📌 3.8 Output-Based: Function Hoisting

js

CopyEdit

sayHello();

function sayHello() {

console.log("Hello");

}

✅ **Output**: "Hello"  
🧠 Because function declarations are hoisted.

## 📌 3.9 Output-Based: Arrow Function Not Hoisted

js

CopyEdit

sayHi(); // Error!

const sayHi = () => {

console.log("Hi");

};

✅ **Output**: ReferenceError  
🧠 Because it's a function expression.

## 📌 3.10 Functions can be assigned, passed & returned

✅ **Example**:

js

CopyEdit

function greet() {

return function() {

console.log("Hi from inner");

}

}

const inner = greet();

inner(); // "Hi from inner"

✅ Common concept in functional programming.

### 🧠 Flash Interview Q Recap:

| **Question** | **Expected Answer** |
| --- | --- |
| What is a function? | A reusable block of code |
| Declared vs Expression? | Declaration is hoisted, Expression is not |
| Arrow function? | Short syntax for function expressions |
| What is an IIFE? | Immediately invoked function |
| Can functions return functions? | Yes |

### 🔍 Mini Challenge

What will be the output?

js

CopyEdit

const greet = function() {

console.log("Hi");

};

function greet() {

console.log("Hello");

}

greet();

✅ Try to guess before scrolling...

<details> <summary>👉 Answer</summary>

**"Hello"**  
Because function declarations are hoisted above expressions. So function greet() overrides const greet = function().

</details>

**⚡️ Module 4: Scope & Closures (JavaScript Interview Prep – Fast & Focused)**

**✅ Q1. What is scope in JavaScript?**

Scope is the context in which variables are accessible.

* **Global Scope** – Accessible everywhere.
* **Function Scope** – Accessible inside a function.
* **Block Scope (let/const)** – Accessible inside {} only.

js

CopyEdit

let a = 10;

function show() {

let b = 20;

console.log(a); // ✅

console.log(b); // ✅

}

console.log(b); // ❌ Error

**✅ Q2. var vs let vs const – Scope difference?**

| **Keyword** | **Scope Type** | **Redeclarable** | **Reassignable** |
| --- | --- | --- | --- |
| var | Function | ✅ Yes | ✅ Yes |
| let | Block | ❌ No | ✅ Yes |
| const | Block | ❌ No | ❌ No |

js

CopyEdit

{

var x = 5;

let y = 10;

}

console.log(x); // 5

console.log(y); // ❌ Error

**✅ Q3. What is a Closure?**

A function **remembering** its **outer scope** even after the outer function has finished executing.

js

CopyEdit

function outer() {

let count = 0;

return function inner() {

count++;

console.log(count);

}

}

const counter = outer();

counter(); // 1

counter(); // 2

👉 Used in private variables & encapsulation.

**✅ Q4. Output-based question on closure:**

js

CopyEdit

function makeCounter() {

let count = 0;

return function () {

return ++count;

};

}

const counter1 = makeCounter();

console.log(counter1()); // ?

console.log(counter1()); // ?

✅ Output:

CopyEdit

1

2

**✅ Q5. Lexical Scope?**

Inner functions can access variables from their outer functions, even if not passed directly.

js

CopyEdit

function outer() {

let name = "JS";

function inner() {

console.log(name); // has access due to lexical scope

}

inner();

}

outer();

**🧠 Mini Recap:**

* Scope = where variable lives
* let/const = block scoped
* Closure = inner function "remembers"
* Use closures for state & data privacy

# ⚡️ ****Module 5: Arrays & Loops (JS Interview Prep – Freshers Focused)****

This is **the most asked** area for practical output questions, especially in service-based companies.

### ✅ Q1. What is an array in JavaScript?

An array is a data structure that holds multiple values in a single variable.

js

CopyEdit

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

### ✅ Q2. How do you loop through an array?

#### 🧪 Example:

js

CopyEdit

const arr = [1, 2, 3];

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

console.log(arr[i]);

}

✅ **Also Asked**:

* for...of
* forEach

js

CopyEdit

arr.forEach(val => console.log(val)); // Cleaner

### ✅ Q3. Output-based question (most expected):

js

CopyEdit

const arr = [1, 2, 3];

arr[5] = 99;

console.log(arr.length); // ?

✅ Output: 6

JS arrays are sparse – it fills missing indexes with undefined.

### ✅ Q4. How do you add/remove items in arrays?

| **Action** | **Method** |
| --- | --- |
| Add to end | push() |
| Add to start | unshift() |
| Remove last | pop() |
| Remove first | shift() |

js

CopyEdit

let nums = [1, 2, 3];

nums.push(4); // [1, 2, 3, 4]

nums.pop(); // [1, 2, 3]

### ✅ Q5. Difference between map, forEach, and filter?

#### forEach() – just iterates:

js

CopyEdit

arr.forEach((num) => console.log(num));

#### map() – returns a new array:

js

CopyEdit

const squared = arr.map(n => n \* n);

#### filter() – returns filtered array:

js

CopyEdit

const even = arr.filter(n => n % 2 === 0);

### ✅ Q6. Most expected output-based Q (filter/map)

js

CopyEdit

const arr = [1, 2, 3, 4];

const result = arr.filter(x => x > 2).map(x => x \* 2);

console.log(result);

✅ Output: [6, 8]

### ✅ Q7. Flatten nested array

js

CopyEdit

const nested = [1, [2, [3]]];

console.log(nested.flat(2)); // [1, 2, 3]

### ✅ Q8. Remove duplicates from array

js

CopyEdit

const nums = [1, 2, 2, 3];

const unique = [...new Set(nums)];

console.log(unique); // [1, 2, 3]

### 🧠 Recap:

| **Concept** | **Key Point** |
| --- | --- |
| forEach | Just loops |
| map | Transforms & returns |
| filter | Filters & returns |
| Set | Removes duplicates |

# ⚡️ Module 6: ****Objects & JSON**** (Freshers Interview Edition – Straight to the Point)

### ✅ Q1. What is an object in JavaScript?

An object is a collection of **key-value** pairs.

js

CopyEdit

const person = {

name: "John",

age: 22,

};

### ✅ Q2. How do you access object properties?

js

CopyEdit

console.log(person.name); // dot notation

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

### ✅ Q3. Add / Update / Delete properties?

js

CopyEdit

person.city = "Mumbai"; // add

person.age = 23; // update

delete person.name; // delete

### ✅ Q4. Most expected output-based question:

js

CopyEdit

const obj = { a: 1, b: 2 };

const copy = obj;

copy.a = 10;

console.log(obj.a); // ?

✅ Output: 10

❗ Objects are reference types – both copy and obj point to the **same memory**.

### ✅ Q5. How to clone an object?

js

CopyEdit

const obj = { a: 1 };

const clone = { ...obj }; // ✅ modern way

### ✅ Q6. How to merge two objects?

js

CopyEdit

const a = { x: 1 };

const b = { y: 2 };

const merged = { ...a, ...b }; // { x: 1, y: 2 }

### ✅ Q7. Loop through object keys/values?

js

CopyEdit

for (let key in person) {

console.log(key, person[key]);

}

✅ Object.keys(obj) – returns keys  
✅ Object.values(obj) – returns values  
✅ Object.entries(obj) – returns [key, value] pairs

### ✅ Q8. JSON – What's that?

JSON = JavaScript Object Notation  
Used for **data exchange** (like in APIs)

js

CopyEdit

const jsonStr = '{"name":"Max","age":20}';

const jsObj = JSON.parse(jsonStr); // Convert to object

const backToJson = JSON.stringify(jsObj); // Convert to string

### ✅ Q9. Common JSON-based question:

js

CopyEdit

const user = { name: "Sam" };

const json = JSON.stringify(user);

console.log(typeof json); // ?

✅ Output: "string"

### 🧠 Recap:

| **Task** | **Method** |
| --- | --- |
| Clone object | { ...obj } |
| Merge objects | { ...obj1, ...obj2 } |
| Parse JSON string | JSON.parse() |
| Convert to JSON | JSON.stringify() |

### ✅ Q1. What is a callback?

A function passed as an argument to another function.

js

CopyEdit

function greet(name, cb) {

cb(name);

}

greet("Aman", function(n) {

console.log("Hello " + n);

});

✅ Freshers are often asked: What is a callback? Why use it?  
🧠 Used to handle **asynchronous code** (e.g., after API, DB query)

### ✅ Q2. What is the problem with callbacks?

**Callback Hell** – too many nested callbacks become messy.

js

CopyEdit

doTask1(() => {

doTask2(() => {

doTask3(() => {

console.log("Done");

});

});

});

### ✅ Q3. What is a Promise?

An object that represents the eventual **success or failure** of an asynchronous operation.

js

CopyEdit

let p = new Promise((resolve, reject) => {

let success = true;

success ? resolve("Done") : reject("Error");

});

p.then(msg => console.log(msg))

.catch(err => console.log(err));

### ✅ Q4. Promise Output Question (Common):

js

CopyEdit

console.log("A");

setTimeout(() => {

console.log("B");

}, 0);

console.log("C");

✅ Output:

css

CopyEdit

A

C

B

JS runs sync code first; setTimeout goes to callback queue.

### ✅ Q5. What is async / await?

Cleaner syntax for working with Promises

js

CopyEdit

async function fetchData() {

let result = await fakeAPI();

console.log(result);

}

function fakeAPI() {

return new Promise(res => {

setTimeout(() => res("Data loaded"), 1000);

});

}

fetchData(); // Logs "Data loaded" after 1 second

### ✅ Q6. Can you await outside an async function?

❌ No. You must wrap it in an async function.

js

CopyEdit

await something(); // ❌ Error

### ✅ Q7. Difference: Promise vs Async/Await?

| **Feature** | **Promises** | **Async/Await** |
| --- | --- | --- |
| Syntax | .then().catch() | await keyword |
| Readability | Can become nested | Cleaner |
| Error Handling | .catch() | try/catch block |

### ✅ Q8. Output-Based: async/await

js

CopyEdit

async function demo() {

return "Hello";

}

demo().then(console.log); // ?

✅ Output: "Hello"

Async functions always return a **Promise**

### 🧠 Quick Recap:

| **Concept** | **Summary** |
| --- | --- |
| Callback | Function passed as param |
| Promise | Handles async code (resolve/reject) |
| async/await | Cleaner way to use promises |

# ⚡️ Module 8: ****Callbacks, Promises & Async/Await****

(Freshers JS Interview – Output-Based & Theory Focused)

### ✅ Q1. What is a callback?

A function passed as an argument to another function.

js

CopyEdit

function greet(name, cb) {

cb(name);

}

greet("Aman", function(n) {

console.log("Hello " + n);

});

✅ Freshers are often asked: What is a callback? Why use it?  
🧠 Used to handle **asynchronous code** (e.g., after API, DB query)

### ✅ Q2. What is the problem with callbacks?

**Callback Hell** – too many nested callbacks become messy.

js

CopyEdit

doTask1(() => {

doTask2(() => {

doTask3(() => {

console.log("Done");

});

});

});

### ✅ Q3. What is a Promise?

An object that represents the eventual **success or failure** of an asynchronous operation.

js

CopyEdit

let p = new Promise((resolve, reject) => {

let success = true;

success ? resolve("Done") : reject("Error");

});

p.then(msg => console.log(msg))

.catch(err => console.log(err));

### ✅ Q4. Promise Output Question (Common):

js

CopyEdit

console.log("A");

setTimeout(() => {

console.log("B");

}, 0);

console.log("C");

✅ Output:

css

CopyEdit

A

C

B

JS runs sync code first; setTimeout goes to callback queue.

### ✅ Q5. What is async / await?

Cleaner syntax for working with Promises

js

CopyEdit

async function fetchData() {

let result = await fakeAPI();

console.log(result);

}

function fakeAPI() {

return new Promise(res => {

setTimeout(() => res("Data loaded"), 1000);

});

}

fetchData(); // Logs "Data loaded" after 1 second

### ✅ Q6. Can you await outside an async function?

❌ No. You must wrap it in an async function.

js

CopyEdit

await something(); // ❌ Error

### ✅ Q7. Difference: Promise vs Async/Await?

| **Feature** | **Promises** | **Async/Await** |
| --- | --- | --- |
| Syntax | .then().catch() | await keyword |
| Readability | Can become nested | Cleaner |
| Error Handling | .catch() | try/catch block |

### ✅ Q8. Output-Based: async/await

js

CopyEdit

async function demo() {

return "Hello";

}

demo().then(console.log); // ?

✅ Output: "Hello"

Async functions always return a **Promise**

### 🧠 Quick Recap:

| **Concept** | **Summary** |
| --- | --- |
| Callback | Function passed as param |
| Promise | Handles async code (resolve/reject) |
| async/await | Cleaner way to use promises |

Say **next** for  
🧩 **Module 9: DOM Manipulation & Event Handling** – perfect for frontend-based questions  
or jump to **Hoisting + Scopes Round**, **ES6+, Spread, Destructuring**, or **Mini Coding Tasks**.

Top of Form



Search

Reason

Bottom of Form