100 JavaScript Interview Questions

With Answers, Code Examples & Real-Life Explanations

📘 Introduction

Welcome to the **“Top 100 JavaScript Interview Questions with Answers, Code Examples & Real-Life Explanations”** — your ultimate guide to mastering JavaScript for interviews and real-world development.

Whether you're preparing for a frontend role, full-stack interview, or just brushing up on key JavaScript concepts, this PDF is structured to help you succeed. Each question includes a clear explanation, relevant code example, and insights that go beyond memorization — helping you understand **why** things work, not just **how**.

These questions are carefully selected from real interview experiences, covering:

- Core JavaScript fundamentals
- Modern ES6+ syntax and features
- Asynchronous JavaScript (Promises, async/await)
- Advanced topics like closures, hoisting, scope, memory, and performance
- Hands-on practical examples used in real applications

JavaScript Interview Questions Index (1–100)

1. What is JavaScript?
2. Difference between var, let, and const?
3. What are arrow functions?
4. What is a closure?
5. Difference between == and ===?
6. What is the DOM?
7. What is hoisting?
8. What are template literals?
9. What is the difference between undefined and null?
10. How does setTimeout work?
11. What are higher-order functions?
12. What is the use of bind(), call(), and apply()?
13. What is event bubbling and capturing?
14. What is the spread operator (...) in JavaScript?
15. What is the rest operator in JavaScript?
16. What are callback functions?
17. Difference between synchronous and asynchronous code?
18. What is Promise in JavaScript?
19. What is async/await?
20. Difference between map(), forEach(), filter()?
21. What is an IIFE (Immediately Invoked Function Expression)?
22. What is lexical scoping?
23. What is the this keyword?
24. What is the difference between stack and heap?
25. How to clone an object in JavaScript?
26. What is event delegation in JavaScript?
27. Difference between function declaration and expression?
28. What is a pure function?
29. Difference between slice and splice?
30. What is the event loop?
31. What is a promise chain?
32. What is optional chaining (?.)?
33. What is nullish coalescing (??)?
34. Difference between synchronous and asynchronous?
35. What is a thunk?
36. Difference between for...of and for...in?
37. What is memoization?
38. What is debouncing?
39. What is throttling?
40. What are data attributes in HTML/JS?
41. What are the different types of errors in JS?
42. How to handle errors in JS?

43. What is this in JavaScript?
44. How does bind() work?
45. Difference between call() and apply()?
46. What is a generator function?
47. What is the new keyword in JS?
48. What are template engines?
49. Difference between innerHTML and textContent?
50. What is strict mode?
51. What is Object.freeze()?
52. What is a Symbol?
53. What are rest parameters?
54. How to clone an object?
55. Difference between deep and shallow copy?
56. What is typeof null?
57. What are falsy values?
58. How to check if a variable is an array?
59. How does JSON.parse() and JSON.stringify() work?
60. What is a tagged template literal?
61. What are JavaScript modules?
62. What is a WeakMap?
63. What is function currying?
64. What are default parameters?
65. How do you compare objects in JS?
66. What is prototype chaining?
67. Difference between == and Object.is()?
68. What is the call stack?
69. What is the microtask queue?
70. What is destructuring assignment?
71. What are optional parameters?
72. What is the role of "use strict"?
73. How to convert a string to number?
74. How to remove duplicates from array?
75. What is .reduce() used for?
76. What is the instanceof operator?
77. What is Promise.all()?
78. What is a ternary operator?
79. What is destructuring with aliasing?
80. How to flatten an array?
81. What is dynamic typing?
82. What is a factory function?
83. What is lexical scope?
84. What is a polyfill?
85. Difference between Object.seal() and Object.freeze()?
86. What are global variables?

87. How to prevent object modification?
88. What is callback hell?
89. What is a higher-order component (HOC)?
90. How to generate random numbers?
91. What is NaN and how to check it?
92. What are template literals?
93. How to merge arrays?
94. What is a constructor function?
95. What is a default case in switch?
96. What is recursion?
97. How to compare arrays?
98. What is .bind() method?
99. What is a WeakSet?
100. How do modules work in JS?

✅ JavaScript Interview Questions

---

1. What is JavaScript?

Answer: JavaScript is a high-level, interpreted programming language used to make web pages interactive. It's a core technology of the web alongside HTML and CSS.

🔸 **Example:**

js

```
alert('Hello, world!');
```

📘 **Explanation:**
This code creates a popup alert with the message "Hello, world!" in the browser.

---

2. Difference between var, let, and const?

Answer:

- `var` is function-scoped.
- `let` and `const` are block-scoped.
- `const` cannot be reassigned.

🔸 **Example:**

js

```
var x = 10;
let y = 20;
const z = 30;
```

📘 **Explanation:**
`x` can be used across functions. `y` and `z` are limited to their block. `z` is constant.

---

3. What are arrow functions?

Answer: Arrow functions are a concise way to write functions. They do not bind their own `this`.

🔸 **Example:**

js

```
const greet = name => `Hello, ${name}`;
console.log(greet('Rachit')); // Hello, Rachit
```

📘 **Explanation:**
This function takes a `name` and returns a greeting using template literals.

---

4. What is a closure?

Answer: A closure is a function that retains access to its lexical scope even after the outer function has returned.

🔸 **Example:**

js

```
function outer() {
  let count = 0;
  return function inner() {
    return ++count;
  };
}
const counter = outer();
console.log(counter()); // 1
```

📘 **Explanation:**
`inner()` retains access to `count` even after `outer()` finishes.

---

5. Difference between == and ===?

Answer:

- `==` compares values after type conversion.
- `===` compares value and type (strict equality).

🔸 **Example:**

js

```
'5' == 5   // true
'5' === 5  // false
```

📘 **Explanation:**
`==` converts types. `===` requires both type and value to match.

---

6. What is hoisting?

Answer: JavaScript hoists declarations (not initializations) to the top of their scope.

🔸 **Example:**

js

```
console.log(a); // undefined
var a = 5;
```

📘 **Explanation:**
The variable `a` is hoisted and initialized as `undefined`.

---

7. What is NaN?

Answer: NaN stands for "Not-a-Number". It represents an invalid number operation.

🔸 **Example:**

js

```
let a = 'abc' / 2;
console.log(a); // NaN
```

📘 **Explanation:**
You can’t divide a string by a number, resulting in NaN.

---

8. What is a callback function?

Answer: A callback is a function passed as an argument to another function.

🔸 **Example:**

js

```
function greet(callback) {
  console.log('Hi');
  callback();
}
greet(() => console.log('Bye'));
```

📘 **Explanation:**
The callback runs after `'Hi'` is printed.

---

9. What are template literals?

Answer: Template literals are string literals using backticks `` ` ``. They support embedded expressions.

🔸 **Example:**

js

```
let name = 'Rachit';
console.log(`Hello, ${name}!`);
```

📘 **Explanation:**
`${name}` is replaced with the variable's value.

---

10. What is an IIFE?

Answer: Immediately Invoked Function Expression — a function that runs as soon as it's defined.

🔸 **Example:**

js

```
(function() {
  console.log('IIFE');
})();
```

📘 **Explanation:**
Helps avoid polluting the global scope.

---

11. What is the difference between null and undefined?

Answer:

- `null` is an assigned value meaning "no value".
- `undefined` means a variable has been declared but not assigned.

🔸 **Example:**

js

```
let a;
let b = null;
console.log(a); // undefined
console.log(b); // null
```

📘 **Explanation:**
Use `null` intentionally. `undefined` often results from missing assignment.

---

12. What is the use of setTimeout()?

Answer: Executes code after a delay.

🔸 **Example:**

js

```
setTimeout(() => console.log('Delayed'), 1000);
```

📘 **Explanation:**
Prints after 1 second.

---

13. Explain event bubbling.

Answer: Events propagate from child to parent elements.

🔸 **Example:**

js

```
document.getElementById("child").addEventListener("click", () => {
  console.log("Child clicked");
});
document.getElementById("parent").addEventListener("click", () => {
  console.log("Parent clicked");
});
```

📘 **Explanation:**
Clicking `child` triggers both handlers.

---

14. What are Promises?

Answer: Promises represent a future value — a wrapper for asynchronous operations.

🔸 **Example:**

js

```
let p = new Promise((resolve, reject) => {
  resolve("Success");
```

```
});
p.then(result => console.log(result));
```

📘 **Explanation:**
`resolve` passes value to `.then()` callback.

---

15. What is async/await?

Answer: Syntactic sugar over Promises for cleaner async code.

🔸 **Example:**

js

```
async function getData() {
  let res = await fetch('/api');
  let data = await res.json();
}
```

📘 **Explanation:**
`await` pauses execution until the promise resolves.

---

16. What are higher-order functions?

Answer: Functions that take other functions as arguments or return functions.

🔸 **Example:**

js

```
const double = n => n * 2;
const apply = (fn, x) => fn(x);
console.log(apply(double, 4)); // 8
```

📘 **Explanation:**
`apply` uses `double` function as input.

---

17. What is destructuring?

Answer: Extracting values from arrays or objects into variables.

🔸 **Example:**

js

```
let [a, b] = [1, 2];
let {name} = {name: 'Rachit', age: 25};
```

📘 **Explanation:**
`a = 1`, `b = 2`, and `name = 'Rachit'`.

---

18. Difference between map and forEach?

Answer:

- `map` returns a new array.
- `forEach` doesn't return anything.

🔸 **Example:**

js

```
[1,2,3].map(x => x * 2);    // [2,4,6]
[1,2,3].forEach(x => x * 2); // undefined
```

📘 **Explanation:**
Use `map` when transformation is needed.

---

19. What is the spread operator?

Answer: `...` expands elements from an iterable.

🔸 **Example:**

js

```
let arr = [1, 2];
let newArr = [...arr, 3]; // [1, 2, 3]
```

📘 **Explanation:**
Combines arrays or copies objects easily.

---

20. What is typeof operator?

Answer: Returns the data type of a value.

🔸 **Example:**

js

```
typeof 'Hello'; // 'string'
typeof 5;       // 'number'
```

📘 **Explanation:**
Useful for type checking.

---

21. What is an object in JS?

Answer: A collection of key-value pairs.

🔸 **Example:**

js

```
let user = {name: 'Rachit', age: 25};
```

📘 **Explanation:**
Access properties with `user.name`.

---

22. How to reverse a string in JavaScript?

Answer: Convert to array, reverse, join.

🔸 **Example:**

js

```
let str = 'hello';
let reversed = str.split('').reverse().join('');
```

📘 **Explanation:**
`split` → `reverse` → `join` gives `olleh`.

---

23. What is JSON?

Answer: JavaScript Object Notation – a format for exchanging data.

🔸 **Example:**

js

```
let obj = {name: 'Rachit'};
let str = JSON.stringify(obj);
```

📘 **Explanation:**
Converts object to JSON string.

---

24. What is the use of localStorage?

Answer: Stores data in the browser with no expiration.

🔸 **Example:**

js

```
localStorage.setItem('name', 'Rachit');
```

📘 **Explanation:**
Use to persist user settings.

---

25. What is the DOM?

Answer: Document Object Model — represents page structure in a tree.

🔸 **Example:**

js

```
document.getElementById('main').innerText = 'Updated';
```

📘 **Explanation:**
Allows JS to update page content.

---

26. What is event delegation in JavaScript?

Answer:
A technique where a single event listener is added to a parent element to manage events for its children.

🔸 **Example:**

js

```
document.getElementById('list').addEventListener('click', function(e)
{
  if (e.target.tagName === 'LI') {
    console.log('Item clicked:', e.target.textContent);
  }
});
```

📘 **Explanation:**
Instead of adding click events to each `<li>`, we add one to the parent `<ul>` and detect clicks on its children.

---

27. What is the difference between function declaration and function expression?

Answer:

- Function declaration is hoisted.
- Function expression is not.

🔸 **Example:**

js

```
greet(); // Works
function greet() {
  console.log('Hi');
}

sayHi(); // Error
const sayHi = function() {
  console.log('Hello');
};
```

📘 **Explanation:**
Only function declarations are hoisted.

---

28. What is a pure function?

Answer:
A function that produces no side effects and returns the same output for the same input.

🔸 **Example:**

js

```
function add(a, b) {
  return a + b;
}
```

📘 **Explanation:**
It doesn't modify anything outside and depends only on parameters.

---

29. Explain the difference between slice and splice.

Answer:

- `slice()` returns a shallow copy without modifying the original array.
- `splice()` modifies the array in place.

🔸 **Example:**

js

```
let arr = [1, 2, 3, 4];
console.log(arr.slice(1, 3));  // [2, 3]
arr.splice(1, 2);              // arr is [1, 4]
```

📘 **Explanation:**
Use `slice` to copy, `splice` to change.

---

30. What is the event loop in JavaScript?

Answer:
The mechanism that handles execution of multiple chunks of code like callbacks, events, etc.

🔸 **Example:**

js

```
console.log('Start');
setTimeout(() => console.log('Timeout'), 0);
console.log('End');
```

📘 **Explanation:**
Even with 0 ms delay, `setTimeout` runs after synchronous code due to the event loop.

---

31. What is a promise chain?

Answer:
Chaining `.then()` methods together to run promises sequentially.

🔸 **Example:**

js

```
fetch('/api')
  .then(res => res.json())
  .then(data => console.log(data));
```

📘 **Explanation:**
Each `then` gets the result of the previous one.

---

32. What is optional chaining (`?.`) in JS?

Answer:
Allows safe access to deeply nested properties.

🔸 **Example:**

js

```
let user = {};
console.log(user?.profile?.email); // undefined, no error
```

📘 **Explanation:**
Avoids runtime errors if `profile` is undefined.

---

33. What is nullish coalescing (`??`)?

Answer:
Returns the right-hand side if the left is `null` or `undefined`.

🔸 **Example:**

js

```
let name = null;
console.log(name ?? 'Guest'); // 'Guest'
```

📘 **Explanation:**
Use when default value is needed only for nullish values.

---

34. Difference between synchronous and asynchronous?

Answer:

- Synchronous: tasks run sequentially.
- Asynchronous: tasks can run in the background.

🔸 **Example:**

js

```
console.log('1');
setTimeout(() => console.log('2'), 0);
console.log('3');
```

📘 **Explanation:**
Output: 1, 3, 2 — setTimeout is async.

---

35. What is a thunk?

Answer:
A function that returns another function, used in Redux middleware for async logic.

🔸 **Example:**

js

```
const thunk = () => dispatch => {
  dispatch({ type: 'LOADING' });
};
```

📘 **Explanation:**
Thunk delays the execution of an action.

---

36. What is the difference between for...of and for...in?

Answer:

- `for...in` iterates over object keys.
- `for...of` iterates over iterable values (arrays, strings).

🔸 **Example:**

js

```
for (let key in {a:1, b:2}) console.log(key); // a, b
for (let val of [1,2]) console.log(val);     // 1, 2
```

📘 **Explanation:**
Choose based on what you're looping through.

---

37. What is memoization?

Answer:
Caching the result of expensive function calls.

🔸 **Example:**

js

```
let cache = {};
function fib(n) {
  if (n in cache) return cache[n];
```

```
  if (n <= 1) return n;
  return cache[n] = fib(n-1) + fib(n-2);
}
```

📘 **Explanation:**
Avoids recalculating previously solved subproblems.

---

38. What is debouncing in JavaScript?

Answer:
Limits function execution by waiting until a delay has passed.

🔸 **Example:**

js

```
function debounce(fn, delay) {
  let timer;
  return function() {
    clearTimeout(timer);
    timer = setTimeout(fn, delay);
  };
}
```

📘 **Explanation:**
Prevents a function from firing too often (like search input).

---

39. What is throttling?

Answer:
Ensures a function runs only once every set interval.

🔸 **Example:**

js

```
function throttle(fn, limit) {
  let lastCall = 0;
```

```
  return function() {
    if (Date.now() - lastCall > limit) {
      lastCall = Date.now();
      fn();
    }
  };
}
```

📘 **Explanation:**
Useful for scroll or resize events.

---

40. What are data attributes in HTML/JS?

Answer:
Custom attributes prefixed with `data-` used to store extra information.

🔸 **Example:**

html

```
<div data-user-id="123"></div>
```

js

```
document.querySelector('div').dataset.userId; // "123"
```

📘 **Explanation:**
Allows storing custom info on DOM elements.

---

41. What are the different types of errors in JS?

Answer:

- SyntaxError
- ReferenceError

- TypeError

🔸 **Example:**

js

```
// ReferenceError
console.log(x); // x is not defined
```

📘 **Explanation:**
Different errors indicate different issues.

---

42. How to handle errors in JS?

Answer:
Using `try...catch` block.

🔸 **Example:**

js

```
try {
  throw new Error("Something went wrong");
} catch (e) {
  console.log(e.message);
}
```

📘 **Explanation:**
Catches runtime errors gracefully.

---

43. What is `this` in JavaScript?

Answer:
Refers to the current execution context.

🔸 **Example:**

js

```
const obj = {
  name: 'Rachit',
  getName() {
    return this.name;
  }
};
```

📘 **Explanation:**
`this` refers to `obj` inside `getName`.

---

44. How does `bind()` work?

Answer:
Returns a new function with a bound `this`.

🔸 **Example:**

js

```
let greet = function() { console.log(this.name); };
let person = { name: 'Rachit' };
let bound = greet.bind(person);
bound(); // Rachit
```

📘 **Explanation:**
Ensures `this` inside `greet` always refers to `person`.

---

45. What is the difference between `call()` and `apply()`?

Answer:
Both invoke functions with a specific `this`, but `call` uses arguments list, `apply` uses array.

🔸 **Example:**

js

```
function greet(msg) {
  console.log(msg + this.name);
}
let user = { name: 'Rachit' };
greet.call(user, 'Hello ');
greet.apply(user, ['Hi ']);
```

📘 **Explanation:**
Use whichever is convenient for your argument format.

---

46. What is a generator function?

Answer:
A function that can pause and resume using `yield`.

🔸 **Example:**

js

```
function* gen() {
  yield 1;
  yield 2;
}
const g = gen();
console.log(g.next().value); // 1
```

📘 **Explanation:**
`yield` returns value and pauses the function.

---

47. What is the `new` keyword in JS?

Answer:
Used to create instances from constructor functions.

🔸 **Example:**

js

```
function Person(name) {
  this.name = name;
}
let p = new Person('Rachit');
```

📘 **Explanation:**
`new` sets up a new object and binds `this`.

---

48. What are template engines in JS?

Answer:
Tools to render HTML with dynamic content (e.g., EJS, Handlebars).

🔸 **Example (EJS):**

html

```
<h1><%= name %></h1>
```

📘 **Explanation:**
Injects `name` variable into HTML.

---

49. What is the difference between innerHTML and textContent?

Answer:

- `innerHTML` parses HTML tags.
- `textContent` just sets plain text.

🔸 **Example:**

js

```
el.innerHTML = "<b>Hello</b>";
el.textContent = "<b>Hello</b>";
```

📘 **Explanation:**
Use `textContent` to avoid HTML injection.

---

50. What is strict mode in JS?

Answer:
A way to catch common coding mistakes by enabling `"use strict"`.

🔸 **Example:**

js

```
"use strict";
x = 10; // ReferenceError: x is not defined
```

📘 **Explanation:**
Prevents using undeclared variables, etc.

---

51. What is `Object.freeze()`?

Answer:
Prevents modification of existing properties and prevents adding/removing properties.

🔸 **Example:**

js

```
const obj = Object.freeze({ name: 'Rachit' });
obj.name = 'New'; // Won't change
```

📘 **Explanation:**
`Object.freeze()` locks the object completely.

---

52. What is a Symbol in JavaScript?

Answer:
A primitive and unique data type used as object property keys.

🔸 **Example:**

js

```
const sym = Symbol('id');
let user = {};
user[sym] = 123;
```

📘 **Explanation:**
Symbols ensure that property keys won't collide.

---

53. What are rest parameters?

Answer:
Allows a function to accept an indefinite number of arguments as an array.

🔸 **Example:**

js

```
function sum(...nums) {
  return nums.reduce((a, b) => a + b);
}
```

📘 **Explanation:**
`...nums` collects arguments into an array.

---

54. How to clone an object?

Answer:
Using `Object.assign()` or spread operator.

🔸 **Example:**

js

```
let clone = { ...original };
```

📘 **Explanation:**
Creates a shallow copy of the object.

---

55. What is the difference between deep and shallow copy?

Answer:

- Shallow copy copies references of nested objects.
- Deep copy copies everything recursively.

🔸 **Example:**

js

```
let obj = { a: { b: 1 } };
let shallow = { ...obj };
obj.a.b = 2;
console.log(shallow.a.b); // 2
```

📘 **Explanation:**
Shallow copy still shares nested objects.

---

56. What is the `typeof null`?

Answer:
It returns `'object'`, which is a known bug in JavaScript.

🔸 **Example:**

js

```
console.log(typeof null); // 'object'
```

📘 **Explanation:**
This is a quirk due to legacy reasons in JS.

---

57. What are falsy values in JS?

Answer:
Values that evaluate to `false` in Boolean context.

🔸 **Falsy Values:**

- `false`, `0`, `''`, `null`, `undefined`, `NaN`

📘 **Explanation:**
All other values are truthy.

---

58. How to check if a variable is an array?

Answer:
Use `Array.isArray()`.

🔸 **Example:**

js

```
Array.isArray([1, 2, 3]); // true
```

📘 **Explanation:**
Avoids confusion with `typeof` which returns `'object'`.

---

59. How does `JSON.parse()` and `JSON.stringify()` work?

Answer:
`parse()` turns JSON string to object; `stringify()` turns object to JSON string.

🔸 **Example:**

js

```
let obj = JSON.parse('{"name":"Rachit"}');
let str = JSON.stringify(obj);
```

📘 **Explanation:**
Used for data interchange or storage.

---

60. What is a tagged template literal?

Answer:
A function that processes a template literal.

🔸 **Example:**

js

```
function tag(strings, name) {
  return strings[0] + name.toUpperCase();
}
tag`Hello ${'rachit'}`; // "Hello RACHIT"
```

📘 **Explanation:**
Provides custom string formatting.

---

61. What are JavaScript modules?

Answer:
Files that export/import functionality using `export` and `import`.

🔸 **Example:**

js

```
// utils.js
export const add = (a, b) => a + b;

// app.js
import { add } from './utils.js';
```

📘 **Explanation:**
Encapsulates code and promotes reusability.

---

62. What is a WeakMap?

Answer:
A Map where keys are only objects and entries are garbage-collected if keys are no longer referenced.

🔸 **Example:**

js

```
let wm = new WeakMap();
let obj = {};
wm.set(obj, 'data');
```

📘 **Explanation:**
Useful for memory-efficient key-value mapping.

---

63. What is function currying?

Answer:
Transforming a function that takes multiple arguments into nested functions.

🔸 **Example:**

js

```
function curry(a) {
  return function(b) {
    return a + b;
  };
}
curry(2)(3); // 5
```

📘 **Explanation:**
Useful for partial function application.

---

64. What are default parameters?

Answer:
Set default values for function parameters.

🔸 **Example:**

js

```
function greet(name = 'Guest') {
  return `Hello ${name}`;
}
```

📘 **Explanation:**
Avoids `undefined` when no argument is passed.

---

65. How do you compare objects in JS?

Answer:
Use deep comparison (e.g., JSON method or libraries).

🔸 **Example:**

js

```
JSON.stringify(obj1) === JSON.stringify(obj2);
```

📘 **Explanation:**
Works for flat objects, not good for nested objects with different key order.

---

66. What is prototype chaining?

Answer:
Objects inherit properties from other objects via the prototype chain.

🔸 **Example:**

js

```
function Person() {}
Person.prototype.greet = function() {
  return 'Hi';
};
let p = new Person();
```

📘 **Explanation:**
`p` inherits from `Person.prototype`.

---

67. What is the difference between `==` and `Object.is()`?

Answer:
`Object.is()` is more accurate for edge cases like `NaN`.

🔸 **Example:**

js

```
NaN == NaN // false
Object.is(NaN, NaN) // true
```

📘 **Explanation:**
`Object.is()` compares more precisely.

---

68. What is call stack in JavaScript?

Answer:
A stack data structure used to keep track of function calls.

🔸 **Example:**

js

```
function a() { b(); }
```

```
function b() { c(); }
function c() { console.log('done'); }
a();
```

📘 **Explanation:**
Each function call gets pushed and popped from the stack.

---

69. What is event loop and microtask queue?

Answer:
Event loop handles tasks and microtasks (like `Promise.then()`) before moving to the next task.

🔸 **Example:**

js

```
setTimeout(() => console.log('timeout'), 0);
Promise.resolve().then(() => console.log('microtask'));
```

📘 **Explanation:**
`microtask` runs before `timeout`.

---

70. What is destructuring assignment?

Answer:
Extract values from arrays or objects into variables.

🔸 **Example:**

js

```
let {name} = {name: 'Rachit'};
let [a, b] = [1, 2];
```

📘 **Explanation:**
Shorthand for pulling data out of structures.

---

71. What is optional parameter in functions?

Answer:
Parameters not required in function calls.

🔸 **Example:**

js

```
function greet(name) {
  return `Hello ${name || 'Guest'}`;
}
```

📘 **Explanation:**
Handles missing parameters gracefully.

---

72. What is the role of `use strict`?

Answer:
Prevents use of undeclared variables and other bad practices.

🔸 **Example:**

js

```
"use strict";
x = 10; // Error
```

📘 **Explanation:**
Adds stricter parsing and error handling.

---

73. How to convert a string to a number?

Answer:
Use `Number()`, unary `+`, or `parseInt()`.

🔸 **Example:**

js

```
+'42';          // 42
Number('42');   // 42
parseInt('42'); // 42
```

📘 **Explanation:**
All convert string to number.

---

74. How to remove duplicates from an array?

Answer:
Use `Set`.

🔸 **Example:**

js

```
let unique = [...new Set([1,2,2,3])]; // [1,2,3]
```

📘 **Explanation:**
`Set` only keeps unique values.

---

75. What is the purpose of `Array.prototype.reduce()`?

Answer:
Used to reduce an array to a single value.

🔸 **Example:**

js

```
[1,2,3].reduce((a, b) => a + b, 0); // 6
```

📘 **Explanation:**
Aggregates array elements.

76. What is the `instanceof` operator?

Answer:
Checks if an object is an instance of a particular class or constructor.

🔸 **Example:**

js

```
let arr = [];
console.log(arr instanceof Array); // true
```

📘 **Explanation:**
Returns `true` if the prototype chain includes the constructor.

---

77. What is `Promise.all()`?

Answer:
Takes an array of promises and resolves when all of them are resolved.

🔸 **Example:**

js

```
Promise.all([Promise.resolve(1), Promise.resolve(2)])
  .then(values => console.log(values)); // [1, 2]
```

📘 **Explanation:**
Used to run multiple async tasks in parallel.

---

78. What is a ternary operator?

Answer:
A shortcut for `if-else`.

🔸 **Example:**

js

```
let age = 20;
let canVote = age >= 18 ? 'Yes' : 'No';
```

📘 **Explanation:**
Evaluates a condition and returns one of two values.

---

79. What is object destructuring with aliasing?

Answer:
Assigns property value to a new variable name.

🔸 **Example:**

js

```
const person = { name: 'Rachit' };
const { name: userName } = person;
```

📘 **Explanation:**
`userName` now holds `'Rachit'`.

---

80. How do you flatten a nested array?

Answer:
Use `.flat()` or recursion.

🔸 **Example:**

js

```
[1, [2, [3]]].flat(2); // [1, 2, 3]
```

📘 **Explanation:**
`.flat()` with depth flattens nested arrays.

---

81. What is dynamic typing in JS?

Answer:
Variables can hold values of any type and change types at runtime.

🔸 **Example:**

js

```
let x = 5;
x = "Hello";
```

📘 **Explanation:**
JavaScript is loosely typed.

---

82. What is a factory function?

Answer:
Returns an object, often used instead of classes.

🔸 **Example:**

js

```
function createUser(name) {
  return { name };
}
```

📘 **Explanation:**
No need for `new` or `this`.

---

83. What is lexical scope?

Answer:
A function's scope is defined by its physical location in code.

🔸 **Example:**

js

```
function outer() {
```

```
  let x = 10;
  function inner() {
    console.log(x);
  }
}
```

📘 **Explanation:**
`inner` has access to `x` due to lexical scope.

---

84. What is a polyfill?

Answer:
Code that adds modern features to older browsers.

🔸 **Example:**

js

```
if (!Array.prototype.includes) {
  Array.prototype.includes = function(search) {
    return this.indexOf(search) !== -1;
  };
}
```

📘 **Explanation:**
Implements `includes()` if it doesn't exist.

---

85. What is the difference between `Object.seal()` and `Object.freeze()`?

Answer:

- `seal()` prevents adding/removing props.
- `freeze()` makes properties immutable too.

📘 **Explanation:**
`freeze` is stricter than `seal`.

---

86. What are global variables?

Answer:
Variables declared outside any function, accessible everywhere.

🔸 **Example:**

js

```
var globalVar = "Hi";
```

📘 **Explanation:**
Accessible throughout the code unless shadowed.

---

87. How to prevent modification of an object?

Answer:
Use `Object.freeze()`.

🔸 **Example:**

js

```
const config = Object.freeze({ debug: true });
```

📘 **Explanation:**
Object is now read-only.

---

88. What is a callback hell?

Answer:
Nested callbacks that become hard to read and maintain.

🔸 **Example:**

js

```
doTask1(() => {
  doTask2(() => {
    doTask3(() => {
      // messy!
    });
  });
});
```

📘 **Explanation:**
Leads to spaghetti code. Use Promises or async/await.

---

89. What is a higher-order component (HOC)?

Answer:
A React concept – functions that take a component and return a new one.

📘 **Explanation:**
Used for code reuse (like authentication, logging).

---

90. How to generate random numbers in JS?

Answer:
Use `Math.random()`.

🔸 **Example:**

js

```
let num = Math.floor(Math.random() * 10); // 0 to 9
```

📘 **Explanation:**
Multiply by range and round down.

---

91. What is NaN and how to check it?

Answer:
NaN = Not a Number. Use `isNaN()` or `Number.isNaN()`.

🔸 **Example:**

js

```
isNaN('abc' / 2);         // true
Number.isNaN('abc' / 2);  // true
```

📘 **Explanation:**
`Number.isNaN()` is more reliable.

---

92. What are template literals?

Answer:
Strings using backticks, support interpolation.

🔸 **Example:**

js

```
let name = 'Rachit';
console.log(`Hi, ${name}`);
```

📘 **Explanation:**
Backticks allow inline expressions.

---

93. How to merge two arrays?

Answer:
Use spread or `concat`.

🔸 **Example:**

js

```
let merged = [...arr1, ...arr2];
```

📘 **Explanation:**
Combines arrays into one.

---

94. What is a constructor function?

Answer:
A function used to create instances using `new`.

🔸 **Example:**

js

```
function Car(make) {
  this.make = make;
}
let honda = new Car('Honda');
```

📘 **Explanation:**
Acts like a class.

---

95. What is a default case in switch?

Answer:
Executed if no `case` matches.

🔸 **Example:**

js

```
switch(x) {
  case 1: break;
  default: console.log('No match');
}
```

📘 **Explanation:**
Fallback when no other case applies.

---

96. What is recursion?

Answer:
A function calling itself.

🔸 **Example:**

js

```
function fact(n) {
  if (n <= 1) return 1;
  return n * fact(n - 1);
}
```

📘 **Explanation:**
Base case ends recursion.

---

97. How to compare two arrays?

Answer:
Use `every()` and `length`.

🔸 **Example:**

js

```
function equal(a, b) {
  return a.length === b.length && a.every((v, i) => v === b[i]);
}
```

📘 **Explanation:**
Compare lengths and each value.

---

98. What is `.bind()` method?

Answer:
Returns a new function with a fixed `this`.

🔸 **Example:**

js

```
const greet = function() { console.log(this.name); };
const person = { name: 'Rachit' };
const boundGreet = greet.bind(person);
boundGreet(); // Rachit
```

📘 **Explanation:**
Useful when passing methods as callbacks.

---

99. What is a `WeakSet`?

Answer:
A set that only holds objects, does not prevent garbage collection.

🔸 **Example:**

js

```
let ws = new WeakSet();
let obj = {};
ws.add(obj);
```

📘 **Explanation:**
Useful for temporary data.

---

100. How do modules work in JavaScript?

Answer:
Code is split into files using `export` and `import`.

🔸 **Example:**

js

```
// lib.js
```

```
export const pi = 3.14;

// app.js
import { pi } from './lib.js';
```

📘 **Explanation:**
Promotes separation of concerns and code reuse.
