Complete JavaScript Interview Guide - Essential Questions

---

1. Difference Between Pass by Value and Pass by Reference

✅*Explanation:*

- **Pass by Value**: The function receives a **copy** of the variable.
- **Pass by Reference**: The function receives a **reference** to the actual variable.

🔍 *Example:*

```
// Pass by Value (Primitive types)
let a = 5;
function modifyValue(x) {
  x = 10;
}
modifyValue(a);
console.log(a); // 5

// Pass by Reference (Objects/Arrays)
let obj = { name: 'John' };
function modifyObject(o) {
  o.name = 'Jane';
}
modifyObject(obj);
console.log(obj.name); // Jane
```

💡 *Interview Tip:*

Understanding this helps in debugging and predicting side effects.

---

2. Difference Between map and filter

✅*Explanation:*

- map() transforms each element.
- filter() removes elements based on a condition.

🔍 *Example:*

```
let numbers = [1, 2, 3, 4];
```

```
// map: doubles each value
let doubled = numbers.map(num => num * 2); // [2, 4, 6, 8]

// filter: only even numbers
let evens = numbers.filter(num => num % 2 === 0); // [2, 4]
```

---

3. Difference Between map() and forEach()

✅*Explanation:*

- map() returns a new array.
- forEach() just executes a function for each element, doesn't return anything.

🔍 *Example:*

```
let arr = [1, 2, 3];

// map
let doubled = arr.map(x => x * 2); // [2, 4, 6]

// forEach
arr.forEach(x => console.log(x * 2)); // prints: 2 4 6
```

---

4. Difference Between Pure and Impure Functions

✅*Explanation:*

- **Pure Function**: No side effects, same output for same input.
- **Impure Function**: Has side effects or depends on external state.

🔍 *Example:*

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

// Impure
let counter = 0;
function increment() {
  counter++;
}
```

---

5. Difference Between for-in and for-of

✅*Explanation:*

- for-in iterates over **keys (property names)**.
- for-of iterates over **values** (in arrays, strings, etc).

🔍 *Example:*

```
let arr = ['a', 'b', 'c'];

for (let i in arr) {
  console.log(i); // 0, 1, 2
}

for (let val of arr) {
  console.log(val); // a, b, c
}
```

---

6. Difference Between call(), apply(), and bind()

✅*Explanation:*

All set the this context manually.

- call: calls function immediately with arguments.
- apply: same as call but takes an array of arguments.
- bind: returns a new function with this bound.

🔍 *Example:*

```
let person = { name: 'Alice' };

function greet(greeting) {
  console.log(`${greeting}, ${this.name}`);
}

greet.call(person, 'Hi');        // Hi, Alice
greet.apply(person, ['Hello']); // Hello, Alice

let greetPerson = greet.bind(person);
```

```
greetPerson('Hey');           // Hey, Alice
```

---

7. Key Features of ES6

✅Highlights:

- let and const
- Arrow functions: () => {}
- Template literals: Hello ${name}
- Default parameters
- Destructuring
- Spread & Rest operators
- class syntax
- Promises
- Modules (import, export)

---

8. Spread Operator (...)

✅Explanation:

Expands an iterable into individual elements.

🔍 Example:

```
let arr1 = [1, 2];
let arr2 = [...arr1, 3, 4]; // [1, 2, 3, 4]

let obj1 = { a: 1 };
let obj2 = { ...obj1, b: 2 }; // { a: 1, b: 2 }
```

---

9. Rest Operator (...)

✅Explanation:

Collects remaining arguments into an array.

🔍 Example:

```
function sum(...args) {
  return args.reduce((acc, val) => acc + val, 0);
}
```

```
console.log(sum(1, 2, 3)); // 6
```

---

10. DRY, KISS, YAGNI, SOLID Principles

✅Definitions:

Principle	Meaning	Tip
DRY (Don't Repeat Yourself)	Avoid duplicating code.	Use functions/components.
KISS (Keep It Simple, Stupid)	Prefer simple solutions.	Avoid overengineering.
YAGNI (You Aren’t Gonna Need It)	Don’t add unused features.	Build only what’s needed.
SOLID	5 OOP principles.	Applies to class design.

🔍 SOLID Breakdown:

- **S**: Single Responsibility
- **O**: Open/Closed
- **L**: Liskov Substitution
- **I**: Interface Segregation
- **D**: Dependency Inversion

---

11. What is Temporal Dead Zone (TDZ)?

✅ Explanation:

- The TDZ is the time between the **entering of a scope** and the **declaration** of a let or const variable, where the variable **cannot be accessed**.

🔍 Example:

```
console.log(a); // ❌ ReferenceError
let a = 10;
```

💡 Why?

- The variable is in the scope but **not initialized** until the line with let a = 10.

---

12. Different Ways to Create Objects in JavaScript

✅ **Methods:**

1. Object Literal:

```
let obj = { name: "John" };
```

2. new Object():

```
let obj = new Object();
obj.name = "John";
```

3. Constructor Function:

```
function Person(name) {
  this.name = name;
}
let p = new Person("John");
```

4. ES6 Class:

```
class Person {
  constructor(name) {
    this.name = name;
  }
}
let p = new Person("John");
```

5. Object.create():

```
let proto = { greet() { console.log("Hello"); } };
let obj = Object.create(proto);
```

---

13. Difference Between Object.keys(), Object.values(), and Object.entries()

Method	Returns
Object.keys(obj)	Array of keys

Method	Returns
Object.values(obj)	Array of values
Object.entries(obj)	Array of [key, value] pairs

🔍 **Example:**

```
let obj = { a: 1, b: 2 };
Object.keys(obj);   // ["a", "b"]
Object.values(obj); // [1, 2]
Object.entries(obj);// [["a", 1], ["b", 2]]
```

---

14. Object.freeze() vs Object.seal()

Feature	Object.freeze()	Object.seal()
Add Property	❌ No	❌ No
Remove Property	❌ No	❌ No
Modify Property	❌ No	✅ Yes (if writable)

🔍 **Example:**

```
let obj = { name: "John" };

Object.freeze(obj);
obj.name = "Jane"; // ❌ No change

Object.seal(obj);
obj.name = "Doe"; // ✅ Changes if sealed (but not frozen)
```

---

15. What is a Polyfill in JavaScript?

✅ **Explanation:**

- A polyfill is code (usually JS) that **adds support** for features not supported in older browsers.

🔍 **Example (polyfill for Array.prototype.includes)**:

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

---

16. What is a Generator Function?

✅ **Explanation:**

- Generator functions can **pause** and **resume** execution using yield.

🔍 **Example:**

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

💡 Useful for lazy evaluation, infinite sequences, and async flows.

---

17. What is a Prototype in JavaScript?

✅ **Explanation:**

- Every JS object has a hidden property [[Prototype]], accessible via __proto__.
- Used for **inheritance**.

🔍 **Example:**

```
let obj = { a: 1 };
let newObj = Object.create(obj);
console.log(newObj.a); // 1 (inherited)
```

---

18. What is IIFE (Immediately Invoked Function Expression)?

✅ **Explanation:**

- A function that is executed **immediately after it's defined**.

🔍 **Example:**

```
(function () {
  console.log("IIFE executed");
})();
```

💡 Used to create private scopes (especially before let/const and modules).

---

19. What is CORS (Cross-Origin Resource Sharing)?

✅ **Explanation:**

- A **security feature** in browsers that restricts web pages from making requests to a **different domain** than the one that served the page.

🔍 **Example:**

- Your frontend app on http://localhost:3000 tries to access API on https://api.example.com.

💡 CORS is controlled via response headers like:

```
Access-Control-Allow-Origin: *
```

---

20. Different Data Types in JavaScript

✅ **Primitive Types:**

- string, number, boolean, null, undefined, symbol, bigint

✅ **Non-Primitive (Reference) Types:**

- object (including arrays, functions)

🔍 **Example:**

```
let str = "Hello";      // string
let num = 42;           // number
let isTrue = true;      // boolean
let nothing = null;     // null
let notDefined;         // undefined
let sym = Symbol("id"); // symbol
let big = 1234567890123456789 0n; // bigint
let obj = { name: "John" };      // object
let arr = [1, 2, 3];             // array (object)
```

---

21. Difference Between TypeScript and JavaScript

Feature	JavaScript	TypeScript
Typing	Dynamically typed	Statically typed (supports type annotations)
Compilation	Interpreted by browser	Compiles to JavaScript
Error Checking	Runtime	Compile-time
OOP Support	Basic	Better support (interfaces, enums, etc.)

🔍 **Example (TypeScript)**:

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

---

22. Authentication vs Authorization

Concept	Authentication	Authorization
Meaning	Verifies identity (who are you?)	Grants access (what can you do?)
Example	Login with email & password	Allow access to admin page
Happens When	First (login step)	After authentication

---

23. Difference Between null and undefined

Type	null	undefined
Meaning	Intentional absence of value	Variable declared but not assigned
Type	Object (weirdly)	Undefined

🔍 **Example:**

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

---

24. Output of 3 + 2 + "7"

🔍 **Answer:**

3 + 2 + "7" = "57"

✅ **Explanation**:

- 3 + 2 = 5
- 5 + "7" → 5 is coerced to "5", so "5" + "7" = "57"

---

25. slice() vs splice()

Feature	slice()	splice()
Mutation	Does **not** modify original	**Modifies** original array
Return	Returns a new array	Returns removed items

🔍 **Example:**

```
let arr = [1, 2, 3, 4];

arr.slice(1, 3); // [2, 3]
arr.splice(1, 2); // arr becomes [1, 4], returns [2, 3]
```

---

26. What is Destructuring?

✅ **Explanation**:
Allows unpacking values from arrays or objects into variables.

🔍 **Example (Object):**

```
const user = { name: "Alice", age: 25 };
const { name, age } = user;
```

🔍 **Example (Array):**

```
const [a, b] = [1, 2];
```

---

27. What is setTimeout in JavaScript?

✅ **Explanation**:
Schedules a function to run **after a delay**.

🔍 **Example:**

```
setTimeout(() => {
  console.log("Runs after 2 seconds");
}, 2000);
```

---

28. What is setInterval in JavaScript?

✅ **Explanation**:
Executes a function **repeatedly** at fixed intervals.

🔍 **Example:**

```
setInterval(() => {
  console.log("Repeats every second");
}, 1000);
```

💡 Use clearInterval() to stop it.

---

29. What are Promises in JavaScript?

✅ **Explanation**:
A Promise represents a value that is available **now, later, or never**.

State	Meaning
Pending	Initial state
Fulfilled	Operation successful
Rejected	Operation failed

🔍 **Example:**

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => resolve("Done"), 1000);
});

promise.then(res => console.log(res)); // Done
```

---

30. What is the Call Stack in JavaScript?

✅ **Explanation**:

- It is a **data structure** that tracks function execution.
- Functions are **pushed** to the stack when called, and **popped** when done.

🔍 **Example:**

```
function first() {
  second();
}
function second() {
  console.log("Hello");
}
first(); // Adds `first` → `second` to call stack
```

💡 Call Stack handles **synchronous** operations. For async ones, the **event loop** takes over.

---

31. What is a Closure?

✅ **Explanation:**
A closure is a function that **remembers** the variables from its **lexical scope** even after that scope has closed.

🔍 **Example:**

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    console.log(count);
  };
}
const counter = outer();
counter(); // 1
counter(); // 2
```

💡 Closures are useful for **data encapsulation**, **private variables**, and **memoization**.

---

32. What are Callbacks in JavaScript?

✅ **Explanation:**
A **callback** is a function passed as an argument to another function to be executed **after** an operation finishes.

🔍 **Example:**

```
function greet(name, callback) {
  console.log("Hi " + name);
  callback();
}

greet("Alice", () => {
  console.log("Callback executed!");
});
```

💡 Often used in **asynchronous code** (e.g., event listeners, setTimeout, API requests).

---

33. What are Higher Order Functions (HOFs)?

✅ **Explanation:**
A **Higher Order Function** is a function that:

- Takes another function as an argument, or

- Returns a function

🔍 **Example:**

```
function multiply(factor) {
  return function(num) {
    return num * factor;
  };
}
const double = multiply(2);
console.log(double(5)); // 10
```

💡 Common HOFs: map(), filter(), reduce()

---

34. Difference Between == and ===

Operator	Name	Comparison Type
==	Loose equality	Compares **values**, does type coercion
===	Strict equality	Compares **value + type**

🔍 **Example:**

```
5 == "5"  // true
5 === "5" // false
```

💡 Use === in all modern JS code for type safety.

---

35. Is JavaScript Dynamically Typed or Statically Typed?

✅ **Answer:**
JavaScript is a **dynamically typed** language.

🔍 **Meaning:**
You don’t need to declare variable types explicitly. Types are determined **at runtime**.

```
let x = 5;     // x is a number
x = "hello";   // now x is a string
```

36. Difference Between IndexedDB and SessionStorage

Feature	IndexedDB	SessionStorage
Storage Size	Large (up to hundreds of MB)	Small (~5MB)
Data Persistence	Until explicitly deleted	Until tab is closed
Structure	Object store (NoSQL-like DB)	Key-value string pairs
Use Case	Offline apps, complex data	Temporary session data

37. What are Interceptors?

✅ **Explanation:**
Interceptors are middleware-like functions (commonly in libraries like **Axios**) that intercept **requests** or **responses** before they are handled.

🔍 **Example (Axios):**

```
axios.interceptors.request.use((config) => {
  config.headers.Authorization = "Bearer token";
  return config;
});
```

💡 Useful for adding **auth tokens**, logging, or error handling.

38. What is Hoisting?

✅ **Explanation:**
Hoisting is JavaScript's behavior of **moving declarations to the top** of their scope before code execution.

🔍 **Example:**

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

💡 Only **declarations** (not initializations) are hoisted. let and const are also hoisted but stay in the **Temporal Dead Zone (TDZ)**.

39. Difference Between var, let, and const

Feature	var	let	const
Scope	Function	Block	Block
Re-declaration	✅ Allowed	❌ Not allowed	❌ Not allowed
Re-assignment	✅ Allowed	✅ Allowed	❌ Not allowed
Hoisting	Yes (undefined)	Yes (TDZ)	Yes (TDZ)

🔍 Example:

```
var x = 1;
let y = 2;
const z = 3;
```

💡 Prefer let and const in modern JS. Avoid var.

---

41. Differences between Promise.all, allSettled, any, and race

Method	Resolves When	Rejects When
Promise.all()	All promises resolve	Any promise rejects
Promise.allSettled()	All promises settle (resolve/reject)	Never rejects
Promise.any()	First promise resolves	All promises reject
Promise.race()	First settled (resolve or reject) promise	First one rejects or resolves

🔍 Example:

```
Promise.all([p1, p2]);        // Waits for all to resolve
Promise.allSettled([p1, p2]);  // Waits for all to finish
Promise.any([p1, p2]);        // Resolves on first success
Promise.race([p1, p2]);       // Resolves/rejects on first finished
```

---

42. Limitations of Arrow Functions

✅ Explanation:
Arrow functions have a **lexical this**, meaning they do **not bind their own this**.

Limitation	Description
No this binding	Inherits this from parent scope
Cannot be used as constructors	Will throw error with new keyword
No arguments object	Use rest params instead (...args)
Not suitable as object methods	this may not refer to the object

🔍 **Example:**

```
const obj = {
  name: "JS",
  greet: () => console.log(this.name), // ❌ 'this' is not bound to 'obj'
};
```

---

43. Difference Between find() vs findIndex()

Method	Returns	Stops When
find()	First matching **value**	When value is found
findIndex()	First matching **index**	When value is found

🔍 **Example:**

```
const arr = [5, 10, 15];
arr.find(v => v > 10);      // 15
arr.findIndex(v => v > 10);  // 2
```

---

44. What is Tree Shaking in JavaScript?

✅ **Explanation:**

- **Tree shaking** is a technique used in **bundlers** (like Webpack, Rollup) to **remove unused code** from the final bundle.

💡 It only works for **ES Modules (import/export)**, not CommonJS.

🔍 **Example:**

```
// Only funcA will be included
import { funcA, funcB } from './utils';
funcA();
```

---

45. Difference Between Local Storage and Session Storage

Feature	Local Storage	Session Storage
Persistence	Until manually cleared	Until tab/browser is closed
Storage Limit	~5–10 MB	~5 MB
Accessibility	Across tabs/windows	Per tab

🔍 **Example:**

```
localStorage.setItem("name", "Alice");
sessionStorage.setItem("token", "123abc");
```

---

46. What is eval() in JavaScript?

✅ **Explanation:**
eval() executes a **string of JavaScript code**.

🔍 **Example:**

```
eval("console.log(2 + 2)"); // Outputs: 4
```

⚠️ **Avoid using** eval():

- Security risks (can execute malicious code)
- Performance issues
- Difficult to debug

---

47. Difference Between Shallow Copy and Deep Copy

Copy Type	Description	Nested Objects
Shallow Copy	Copies top-level properties only	References nested objects
Deep Copy	Recursively copies all levels	Clones nested objects too

🔍 **Example:**

```
let obj = { a: 1, b: { c: 2 } };

// Shallow copy
let shallow = { ...obj };
```

```
shallow.b.c = 3;
console.log(obj.b.c); // 3 ❌

// Deep copy
let deep = JSON.parse(JSON.stringify(obj));
deep.b.c = 4;
console.log(obj.b.c); // 3 ✅
```

---

48. Difference Between Undeclared and Undefined Variables

Type	Description
Undeclared	Variable never declared (ReferenceError)
Undefined	Declared but not assigned

🔍 **Example:**

```
console.log(x); // ❌ ReferenceError: x is not defined
let y;
console.log(y); // ✅ undefined
```

---

49. What is Event Bubbling?

✅ **Explanation:**

- In event bubbling, events **propagate from child → parent → topmost**.

🔍 **Example:**

```
<div onclick="console.log('Parent')">
  <button onclick="console.log('Child')">Click</button>
</div>
```

✅ Clicking button logs:

```
Child
Parent
```

💡 You can stop it with event.stopPropagation().

---

50. What is Event Capturing?

✅ **Explanation:**

- In event capturing, events **propagate from parent → child** (opposite of bubbling).

🔍 **Example:**

element.addEventListener('click', handler, true); // `true` enables capture phase

✅ **Propagation order:**

1. Capturing phase (top → target)
2. Target phase
3. Bubbling phase (target → top)

---

51. What are Cookies?

✅ **Explanation:**
Cookies are small pieces of **data stored in the browser** to remember information about the user.

□ **Used for:**

- Session management (e.g., login tokens)
- Personalization (e.g., language settings)
- Tracking (e.g., analytics)

🔍 **Example:**

document.cookie = "username=John; expires=Fri, 18 Jul 2025 12:00:00 UTC; path=/";

💡 Cookies are:

- Limited to ~4KB per cookie
- Sent with **every HTTP request**

---

52. typeof Operator

✅ **Explanation:**
typeof is used to check the **type of a variable or expression**.

🔍 **Examples:**

```
typeof 123          // 'number'
typeof "hello"      // 'string'
typeof true         // 'boolean'
typeof {}           // 'object'
typeof []           // 'object' ❗
typeof null         // 'object' ❗ (JS quirk)
typeof undefined    // 'undefined'
typeof function(){}  // 'function'
```

💡 typeof null returning 'object' is a **known bug** in JavaScript for backward compatibility.

---

53. What is this in JavaScript?

✅ **Explanation:**
this refers to the **object that is executing the current function**.

🔄 **Behaves differently in various contexts:**

Context	this Refers To
Global scope (non-strict)	window (in browsers)
Global scope (strict)	undefined
Object method	That object
Function (non-strict)	window
Function (strict)	undefined
Arrow function	Lexically inherited this
Class method	Class instance
Event handler	DOM element (unless bound)

🔍 **Examples:**

```
function normal() {
  console.log(this);
}
const arrow = () => {
  console.log(this);
};
normal();      // window or undefined
arrow();       // lexical `this`

const obj = {
  name: "JS",
  greet: function() {
    console.log(this.name);
  }
};
obj.greet();   // JS
```

---

54. How Do You Optimize the Performance of a Web Application?

✅ **Answer Overview:**

Frontend optimizations:

- Use **lazy loading** for images/components
- **Minify and bundle** JS/CSS
- Use **code splitting**
- Use **React.memo**, useMemo, useCallback to prevent re-renders
- Compress assets (GZIP, Brotli)
- Use **efficient state management**
- Debounce search inputs or API calls
- Reduce DOM manipulation

Backend optimizations:

- Use **caching** (Redis, in-memory)
- Optimize **database queries**
- Apply **pagination** for large lists
- Use a **CDN** for static content

Tools to monitor:

- Lighthouse
- Chrome DevTools
- Web Vitals

---

55. What is Debouncing and Throttling?

✅ *Debouncing:*

Delays execution **until** after a certain time has passed **since the last event**.

🔍 **Use Case:** Search bar — wait until user stops typing.

```
function debounce(fn, delay) {
  let timer;
  return function (...args) {
    clearTimeout(timer);
    timer = setTimeout(() => fn.apply(this, args), delay);
  };
}
```

✅ *Throttling:*

Ensures a function is called **at most once** every fixed interval, no matter how many times the event fires.

🔍 **Use Case:** Window resize or scroll events.

```
function throttle(fn, limit) {
  let lastCall = 0;
  return function (...args) {
    const now = Date.now();
    if (now - lastCall >= limit) {
      lastCall = now;
      fn.apply(this, args);
    }
  };
}
```
