JavaScript Interview Q&As

🔹 Section 1: Core Concepts

Q1. What is Hoisting in JavaScript?

Answer : Hoisting means JavaScript moves variable and function declarations to the top of their scope before code executes.

- var is hoisted and initialized as undefined.
- let and const are hoisted too, but not initialized (they stay in **Temporal Dead Zone**) until the line of declaration.
  💡 Tip: Always say — "Hoisting affects how we access variables before declaration."

👉 Example with `var`:

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

Here, JS interprets it like:

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

👉 Example with `let/const`:

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

Q2. Difference between var, let, and const?

Answer : var = function-scoped, can be redeclared and updated.

let = block-scoped, can be updated but not redeclared.
const = block-scoped, cannot be updated or redeclared (but objects/arrays can mutate).
💡 Tip: Prefer let and const in modern JS to avoid scope bugs.

Q3. What are Closures?
Answer : A closure is when an inner function "remembers" variables from its outer function, even after the outer function has finished running.
Example:

```js
function counter() {
  let count = 0;
  return function() { count++; return count; }
}
const c = counter();
c(); // 1, still remembers count
```

💡 Tip: Closures are the base of React hooks like useState.

- Functions like setTimeout, event handlers, and useState in React depend on closures.

Q4. Difference between == and ===?
Answer : == compares only value (does type conversion), while === compares both value **and** type.
Example: "5" == 5 → true, "5" === 5 → false.
💡 Tip: Always use === for safer, bug-free comparisons.

Q5. What is Scope in JavaScript?
Answer : Scope decides where variables can be accessed.

- **Global scope** = accessible everywhere.
- **Function scope** = accessible only inside function.
- **Block scope** = inside {} with let/const.
  Answer : Example:

```js
{ let x = 5; }
console.log(x); // ❌ not accessible
```

💡 Tip: "let" and "const" respect block scope → cleaner code.

🔹 Section 2: Functions & Async

Q6. What are Arrow Functions?

Answer : Arrow functions give a shorter syntax and don't create their own this (they use the parent's this).

Example:

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

Good for callbacks, not good for object methods that depend on this.

💡 Tip: Use arrow functions for clean, inline code.

Q7. What is the Event Loop in JS?

Answer : JS is single-threaded but uses an **event loop** to manage async tasks (like setTimeout, promises, or I/O).

It works by moving async tasks to a **callback queue**, then pushes them into the **call stack** when free.

💡 Tip: Classic interview question — "Why does setTimeout run after code finishes?" → Event loop.

Q8. Difference between Callbacks, Promises, and async/await?

Answer : **Callbacks** = function passed into another (leads to "callback hell").

Promises = cleaner way to handle async results (.then, .catch).

async/await = makes async code look synchronous → easier to read.

💡 Tip: Show callback hell vs async/await code in interviews.

Q9. What is Debouncing & Throttling?

Answer : **Debounce** = runs only after user stops triggering an event

(e.g., typing search box).
Throttle = runs at fixed intervals (e.g., scroll or resize events).
Example: Search input uses debounce, infinite scroll uses throttle.
💡 Tip: Both improve performance in UI-heavy apps.

Q10. Explain this keyword in JS.
Answer : this depends on **how a function is called**.

- Global → window (in browsers).
- Inside object method → the object itself.
- Arrow functions → parent's this, not their own.
  💡 Tip: Always mention → "Arrow functions don't bind this."

🔷 Section 3: Objects & Prototypes

Q11. Difference between null and undefined?
Answer: null is an intentional empty value set by the developer.
undefined means a variable has been declared but not assigned any value.
Example:

```
let a; // undefined
```

```
let b = null; // null
```

💡 **Tip**: Always explain → null is intentional, undefined is accidental.

Q12. What are Prototypes in JS?
Answer: Every object in JavaScript has a hidden property called prototype, which is another object that can provide properties and methods.
When you access a property that doesn't exist, JS looks up the **prototype chain**.
Example: Arrays inherit map, filter from Array.prototype.
💡 **Tip**: Say — "Prototypes enable inheritance in JavaScript."

Q13. Difference between Shallow Copy vs Deep Copy?
Answer: Shallow copy only copies the first level and keeps references for nested objects.
Deep copy creates a full independent copy of all levels.
Example:

```
let obj = { a: { b: 1 } };
let shallow = {...obj}; // shares nested reference
let deep = structuredClone(obj); // true copy
```

💡 **Tip**: Always mention → "Shallow copy breaks with nested objects."

Q14. What are Higher-Order Functions (HOFs)?
Answer: A HOF is a function that either takes another function as an argument or returns a function.
Example: map, filter, reduce are common HOFs.
They make code reusable and functional.
💡 **Tip**: Always say — "HOFs improve reusability in modern JS."

Q15. Difference between map, forEach, and reduce?
Answer:

- map returns a new array after transforming each item.
- forEach just loops, doesn't return.
- reduce accumulates values into a single result.
  Example:

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

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

💡 **Tip**: Use map for transformations, reduce for aggregations.

🔹 Section 4: Advanced Concepts

Q16. Difference between call, apply, and bind?
Answer:

- call → invokes a function with arguments given one by one.
- apply → similar to call, but takes arguments as an array.
- bind → doesn't call immediately, instead returns a new function with this bound.
  Example: func.call(obj, a, b), func.apply(obj, [a, b]).
  💡 **Tip**: Always mention → "bind is used for callbacks with fixed this."

Q17. What is Currying in JS?
Answer: Currying transforms a function with multiple parameters into a sequence of functions that take one argument at a time.
Example:

`function sum(a){ return b => c => a+b+c }`

`sum(1)(2)(3); // 6`

It improves reusability and partial function application.
💡 **Tip**: Show quick example → "sum(2)(3) returns 5."

Q18. What is Memoization?
Answer: Memoization is an optimization technique where expensive function results are cached and reused when the same inputs occur again.
Example:

```
function factorial(n, cache={}) {
  if(n in cache) return cache[n];
  return cache[n] = n<=1 ? 1 : n*factorial(n-1, cache);
}
```

💡 **Tip**: Mention → "Memoization improves performance in heavy computations."

Q19. Difference between Shallow vs Deep Equality?
Answer: Shallow equality compares object references, not contents.
Deep equality compares actual nested values.
Example:

```
{} === {} // false (different memory)
```

```
_.isEqual({a:1}, {a:1}) // true (deep equal with lodash)
```

💡 **Tip**: In interviews, always explain — "Shallow checks memory, deep checks structure."

Q20. What is Event Delegation?
Answer: Event delegation means attaching a single event listener on a parent element to handle events for its children (current or future).

Event delegation is a JavaScript technique where you **don’t add event listeners to multiple child elements individually**, but instead, you add **a single event listener to a common parent element**.
That parent “delegates” the event handling by checking which child triggered the event.

📌 Example without delegation

```javascript
// Bad approach: adding event to each button
const buttons = document.querySelectorAll("button");
buttons.forEach(btn => {
  btn.addEventListener("click", () => {
    console.log("Button clicked");
  });
});
```

👉 Problem: If you add new buttons later with JS, they won't have listeners.

📌 Example with Event Delegation

```javascript
// Good approach: add one listener to parent
document.body.addEventListener("click", e => {
  if (e.target.tagName === "BUTTON") {
    console.log("Button clicked:", e.target.innerText);
  }
});
```

👉 Now, **any button** (existing or newly created) will trigger the event.

💡 **Tip**: Always say — "It improves performance in dynamic DOMs."