JavaScript Questions and Answers

1. What is the difference between `let`, `const`, and `var` in JavaScript?

Answer: The difference lies in scope, hoisting, and mutability. `var` is function-scoped, hoisted, and allows redeclaration. `let` is block-scoped and doesn't allow redeclaration. `const` is also block-scoped, but it must be initialized and cannot be reassigned (though object contents can still be modified).

2. Can you reassign a `const` variable if it's an object? Explain with example.

Answer: Yes, you can modify properties of a `const` object. Example:

const obj = { name: 'John' }; obj.name = 'Jane'; // valid. But obj = {} would throw an error.

3. What is block scope in the context of `let` and `const`?

Answer: `let` and `const` are block-scoped, which means they are only accessible within the `{}` they are defined in. This helps prevent unintended variable access or mutation outside intended blocks.

4. What is function hoisting? Give an example.

Answer: `Function hoisting` means you can call a function before its declaration if it's defined with `function` keyword. Example: greet(); function greet() { console.log('Hi'); }

5. How does hoisting differ between variables declared with `var` and `let`?

Answer: `var` is hoisted and initialized as `undefined`, so you can access it before declaration (though not recommended). `let` and `const` are hoisted too, but not initialized, so accessing them before declaration gives a ReferenceError.

6. What’s the difference between a function declaration and a function expression?

Answer: Function declarations are hoisted, meaning they can be used before they are defined. Function expressions (including arrow functions) are not hoisted, so they must be defined before use.

7. What is object destructuring? Give an example.

Answer: Object destructuring lets you extract values from objects into variables. Example:

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

8. How does array destructuring work in JavaScript?

Answer: Array destructuring lets you assign elements of an array to variables in a single line. Example:

const [a, b] = [10, 20];

9. Can you destructure nested objects? Show how.

Answer: Yes. Example:

const obj = { user: { name: 'Tom', age: 30 } }; const { user: { name } } = obj; // name = 'Tom'

10. What is the spread operator `...` used for? Provide an example.

Answer: Spread operator `...` expands iterable elements. Example:

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

11. How is the rest parameter different from the spread operator?

Answer: Rest collects multiple elements into a single parameter. Example:

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

12. How can you use spread to clone an object or array?

Answer: You can clone objects/arrays using spread. Example:

const arr2 = [...arr1]; const obj2 = { ...obj1 };

13. What is a Promise in JavaScript?

Answer: A Promise is an object representing the eventual completion or failure of an async operation. It has three states: pending, fulfilled, or rejected.

14. How does `.then()` work in a Promise chain?

Answer: `.then()` registers callbacks for when a promise resolves. It returns another promise, enabling chaining.

15. How do you handle errors in Promises?

Answer: Errors in promises can be caught using `.catch()` at the end of the promise chain.

JavaScript Questions and Answers

16. What is `async` and `await` used for in JavaScript?

Answer: `async` marks a function as asynchronous and returns a promise. `await` pauses the function execution until the awaited promise resolves.

17. Can you use `await` outside an `async` function?

Answer: No. `await` must be used inside an `async` function or at the top level (in some modern JS environments).

18. How do you handle errors in `async/await`? Show an example.

Answer: Use try-catch to handle errors in `async` functions:

try { await fetchData(); } catch (e) { console.error(e); }

19. What are object methods in JavaScript?

Answer: Object methods are functions stored as object properties. Example:

const user = { greet() { console.log('Hello'); } };

20. How can you add or remove a property from an object dynamically?

Answer: You can use `obj[prop] = value` to add or update, and `delete obj[prop]` to remove properties dynamically.

21. What is the difference between dot notation and bracket notation in accessing object properties?

Answer: Dot notation (`obj.name`) is simple, but bracket notation (`obj['name']`) is useful for dynamic property access or keys with special characters.

22. What is the difference between `onclick` and `addEventListener`?

Answer: `addEventListener` is preferred because it allows multiple listeners and separation of behavior. `onclick` can be overwritten.

23. How do you prevent default behavior of a form submission in JavaScript?

Answer: Use `event.preventDefault()` in a form submit handler to stop default submission and do custom logic.

JavaScript Questions and Answers

24. How can you select and modify DOM elements using JavaScript?

Answer: Use `document.querySelector()` or `getElementById()` to select elements, and `.textContent`, `.innerHTML`, `.classList` to modify them.

25. What is scope in JavaScript?

Answer: Scope defines where a variable is accessible. JavaScript has global, function, and block scope.

26. What is a closure? Explain with a simple example.

Answer: A closure is a function that remembers variables from its outer scope even after that scope has exited. Example:

function outer() { let x = 10; return function() { console.log(x); }; }

27. What is lexical scope in JavaScript?

Answer: Lexical scope means a function can access variables defined in the scope where it was created, not where it's called.

28. What is a higher-order function? Give an example.

Answer: A higher-order function takes other functions as arguments or returns a function. Example: `setTimeout`, `map`, `filter`.

29. How does `map()` work as a higher-order function?

Answer: `map()` is a higher-order function that transforms each element in an array and returns a new array.

30. Explain the difference between `call`, `apply`, and `bind` methods.

Answer: `call()` invokes a function with a given `this` and arguments. `apply()` is the same but takes arguments as an array. `bind()` returns a new function with fixed `this`.

31. How do you add and remove elements from an array in JavaScript?

Answer: Use `.push()` to add to end, `.pop()` to remove from end, `.unshift()` to add to start, `.shift()` to remove from start.

JavaScript Questions and Answers

32. How can you loop through an array?

Answer: You can use `for`, `for...of`, `forEach()`, or higher-order functions like `map()`, `filter()` to loop through arrays.

33. What is the difference between `map()`, `filter()`, and `reduce()`?

Answer: `map()` returns a new array by transforming each item. `filter()` returns items that pass a condition. `reduce()` combines all items into a single result.

34. What is an arrow function? Provide syntax.

Answer: Arrow functions are concise syntax: `(a, b) => a + b`. They're great for short, inline functions.

35. Do arrow functions have their own `this`?

Answer: Arrow functions do not bind their own `this`. Instead, they inherit `this` from the enclosing scope.

36. Can arrow functions be used as constructors?

Answer: No, arrow functions can't be used as constructors because they don't have the internal `[[Construct]]` method.

37. What does `this` refer to in JavaScript?

Answer: `this` refers to the object that owns the current function. In global context, it's `window` (or `undefined` in strict mode).

38. How does `this` behave in strict mode?

Answer: In strict mode, `this` in a function is `undefined` instead of defaulting to the global object.

39. How does `this` differ in arrow functions vs regular functions?

Answer: Regular functions bind `this` based on how they are called. Arrow functions use `this` from where they were defined.

40. How can you explicitly set `this` using `call`, `apply`, or `bind`?

JavaScript Questions and Answers

Answer: `call()` invokes a function with specified `this`. `apply()` does the same but with arguments as an array. `bind()` returns a new function with permanently set `this`.